Application Of Enterprise Identifiers (EID) To Automated Information Systems For Personnel

September 2002

Prepared For:

Mr. Bruce Haberkamp
ATTN: SAIS-IOE
107 Army Pentagon
Washington, DC 20310-0107

Prepared By:

Sam Chamberlain, Ph.D.
Army Research Laboratory
Aberdeen Proving Ground, MD 21005-5067

and

Francisco Loaiza, Ph.D.
Windsor Lin, Ph.D.
Institute for Defense Analyses
1801 N. Beauregard St.
Alexandria, VA 22311

This page left blank
intentionally

TABLE OF CONTENTS

1. Introduction 6
2. Background – Progress Since the Logistics Study 6
2.1 A Short Review of Organization Identifiers and Hierarchical Structures 7
3. Lessons Learned Since The Logistics Study 10
3.1 Decoupling Enterprise Identifiers (EIDs) From Organization Identifiers (Org-IDs) 10
3.1.1 EID Server Architectures 12
3.1.2 EID Seed Servers (ESS) 14
3.1.3 EID Servers 15
3.1.4 EID Server Location Flexibility 16
3.2 EIDs as Alternate Keys 19
3.3 Application Of The EID Concept To Reference Libraries 19
3.4 Data Ownership, Authoritative Sources, and Duplication 21
3.5 User Access To EID Servers 23
3.6 Choice Of Optimal Taxonomies 23
3.7 Benefit To The Warfighter 25
3.8 Benefits To Database Evolution And Managing Change 27
4. Personnel Ontology and Modeling Issues 28
4.1 Basic Entities in Battle Command System Models 28
4.2 Personnel Data in the Context of Battle Command Systems Data Models 29
4.2.1 Organization [ORG] 29
4.2.2 Organization-Type [ORGT] 30
4.2.3 Person [PERS]. 32
4.2.4 Person-Type [PERST] and Skill-Type [SKILLT] 32
4.2.5 Attributes for PERST and SKILLT 35
4.2.6 Modeling Options for Personnel Attributes 38
4.2.7 Using EIDs to Add Flexibility to Associations 43
5. Recommendations and summary. 51
5.1 Recommendation 1: Incorporate EIDs into personnel automated information systems as alternate keys 51
5.2 Recommendation 2: To facilitate interoperability, a special external function called "Fetch-EID" should be included into the interfaces of personnel AIS's along with support for XML. 52

5.3 Recommendation 3: EIDs should be implemented as 64-bit sequences, or alternatively, as 16 characters using hexadecimal representation, using a global EID prefix and a locally generated EID suffix. 52

5.4 Recommendation 4: An initial interface between administrative and battle command systems for personnel data should be established via the Army Organization Server (AOS). 53

5.5 Recommendation 5: PERSCOM should be the authoritative source for Army person-type data maintained in the AOS. 54

5.6 Recommendation 6: To facilitate inter-service exchange, personnel data in the AOS should be modeled using both bundled and generic attributes similar to a meta-model. 54

Annex A: Analysis of Options for EID Implementations 57

Summary 62

I. Basic Concepts And Definitions 67

II. Technical Alternative Analysis 71

LIST OF FIGURES

Figure 1: Tree Graph Definitions and Terms 8
Figure 2: Org-ID Servers Provide ORG-IDs For Organization Servers That Maintain Force Structure Data 11
Figure 3: Original Scheme to Create EIDs from Org-IDs 12
Figure 4: EID Server Architecture 13
Figure 5: Structurally, EIDs Based Upon EID Seeds Are Identical To Those Based Upon Org-IDs 13
Figure 6: EID Server Location Options 17
Figure 7: Extreme Case – Centralized 18
Figure 8: Extreme Case – Decentralized 18
Figure 9: Mixed Extreme One 18
Figure 10: Mixed Extreme Two 18
Figure 11: Suite of Servers that Contain Stationary Data 20
Figure 12: Arbitrary Plug and Play of References to Disparate Entities 26
Figure 13: Examples of Instances and their Associated Templates 29
Figure 14: ORGT and ORG Trees 31
Figure 15: PERST Used to Define Requirements for an ORGT 33
Figure 16: LC2IEDM Associations with Person-Type 33
Figure 17: Person-Type, Skill-Type, Org-Type, and Person Associations 34
Figure 18: Examples of Officer Classification Schemes 36
Figure 19: IDEF1X Representation of Aggregate Occupation Attribute 37
Figure 20: IDEF1X Representation of Decomposed Attributes 40
Figure 21: Generic Attributes in Entity 41
Figure 22: Bundling Attributes into Entities 42
Figure 23: Generic Skill-Type With Bundled Person-Type 43
Figure 24: Association Tables with Generic Attributes 44
Figure 25: Using a Single Entity for Generic Attributes 45
Figure 26: Bundled Attributes and Association Table 46
Figure 27: Bundled Skill-Types 47
Figure 28: Personnel Data Maintained at Resolutions Lower than Position 48
Figure 29: LC2IEDM Establishment (Modified for EIDs) 49
Figure 30: Aggregate Personnel Information Accessed Via ORGT Associations 50

1. Introduction

This study presents an analysis of the ways in which enterprise identifiers (EID) can be exploited to facilitate information exchange between automated information systems (AIS) that handle personnel information. This study addresses several facets of this challenge, to include the selection of modeling techniques that exploit the capabilities of EIDs to: (1) integrate administrative AISs with battle command systems, (2) operate across military service and coalition boundaries, and (3) provide simple maintenance of personnel type information within an integrated Army Organization Server (AOS) that contains comprehensive force structure details. During FY00, an Army Studies Program study was conducted on behalf of the Director for Plans, Operations, and Logistics (BG H.A. Curry) of the U.S. Army Office of the Deputy Chief of Staff for Logistics (ODCSLOG) that investigated the utility of using enterprise identifiers to address issues encountered by the logistics community, and in particular, the building of units "on-the-fly."[1] In the report, eight recommendations were presented. Since that time (and during the performance of this study) new insights have emerged that have changed some of the original implementation concepts of enterprise identifiers (EID) for battlefield and business systems. This report describes those changes and provides recommendations for applying EIDs to personnel systems.

2. Background – Progress Since the Logistics Study

During this study, experts from the personnel automation community were consulted and dialog ensued concerning the use of EIDs in current and planned personnel related systems. The experts were:

Mr. Smokey Bresser	– Architecture & Information Assurance Office, ODCSPER.
Mr. Paul Oestreich	– PERSCOM, PERSINSD TAPC-PSA-SA
Ms. Peggy Mercer	– PERSCOM, PERSINSD TAPC-PSA-SA
Ms. Chris Lundeen	– PERSCOM, PERSINSD TAPC-PSA-SA
LTC Girard Evans	– PERSCOM, PM-ITAPDB
Mr. Michael Monteleone	– Joint Requirements & Integration Office, DUSD-PI, USD-P&R.

Of particular interest are the interactions between administrative systems, which are used in a fixed or non-deployed environment, and battlefield systems that are mobile and may be used in combat. Several schemas are being developed concurrently for the administrative environment. The ultimate goal is for all personnel systems to migrate to a Department of Defense (DoD) standard, and this is one of the objectives of the Defense Integrated Military Human Resources System (DIMHRS) program.[2] In discussions with Mr. Montelone,[3] the use of commercial standards and products will be a key element of DIMHRS. This is also true for the Integrated Total Army Personnel Database (ITAPDB) being developed by the US Total Army Personnel

[1] Enterprise Identifiers For Logistics: *An Approach in Support of Army Transformation Initiatives*, December 2000. See: http://arch-odisc4.army.mil/data_mgt/Org_ID_Task_Docs.asp

[2] DIMHRS - See: http://www.mpm.osd.mil/dimhrs.htm.

[3] Meeting held 1 August 2001 in Arlington, VA.

Command (PERSCOM)[4] to consolidate personnel data from Active Army, Army Reserve, Army National Guard, and Army Civilian Personnel databases. Ultimately, ITAPDB will serve as the data source for migrating data to the DIMHRS.

The Army Battle Command System (ABCS) is a primary example of a combat system that includes personnel data.[5] During battle, it will be the primary source of basic personnel data for commanders and their staffs. ABCS depends upon the Joint Common Database (JCDB)[6] for storing and maintaining information. The details of how personnel data will migrate between administrative systems with a commercial basis and battlefield systems are still being developed. However, a major task will be the routine maintenance and update of information between the systems as conditions change. To accomplish this, it will be necessary to link data that resides in several different systems, with different schemas, based upon different perspectives. This will require data to be explicitly associated at the entity level of detail or higher. Enterprise identifiers significantly facilitate this capability.

2.1 *A Short Review of Organization Identifiers and Hierarchical Structures*

EIDs evolved from a study of organization identifiers (Org-ID).[7] To understand EIDs, it is beneficial to understand the concepts behind Org-IDs. From a personnel perspective, the main concept in this section is that billets are simply a type of organization, and therefore, should have semantics consistent with other force structure properties. Although the current study suggests that EIDs should be managed independent of the force structure community, the close association between the personnel and force structure communities requires a basic understanding of the concepts behind formally representing hierarchical structures when applied to military organizations.

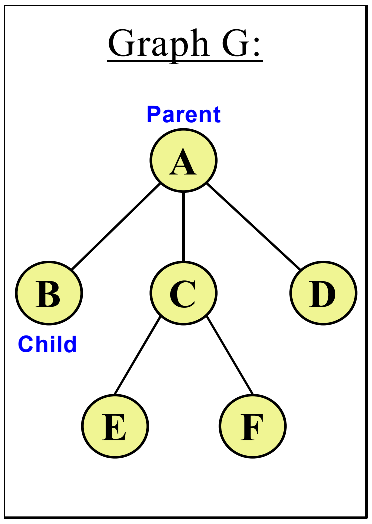
To explicitly represent force structure, tree graphs can be utilized. Although numerous informal interactions are common among military organizations, due to the basic principles of military command, military organizations are conveniently represented via hierarchical organization charts that describe the aggregation and composition of clusters of people and equipment. To formalize the process of building and rapidly modifying organization charts, they can be represented in terms of graph theory. A ***graph*** is composed of a set of ***nodes*** connected by a set of ***links*** (i.e., called vertices {V} and edges {E}, respectively, in mathematical vernacular). A ***tree*** is a special type of graph that is *fully connected* (i.e., every node is linked to at least one other node) and there are *no cycles* (i.e., when links are traversed, only one path exists between any two nodes). Normally, one node is selected as the "beginning" of the tree and is named the root node. Figure 1 summarizes several tree graph terms and illustrates how they are easily exploited to denote hierarchical organization charts.

[4] ITAPDB: See: http://www-perscom.army.mil/persinsd/ITAPDB/ITAPDB - Home Page.htm and https://itapdb.hoffman.army.mil/homesite/Default.htm

[5] In particular, the Combat Service Support Control System (CSSCS), a sub-system of ABCS; See http://www.lee.army.mil/csscs/documentation.htm

[6] The JCDB schema is the JCDBDM: *Joint Common Database Data Model*; See: https://www.kc.us.army.mil/homepeoc3s.nsf/Home?OpenFrameSet

[7] Chamberlain, S., "Default Operational Representations of Military Organizations," *Army Research Laboratory Technical Report: ARL-TR-2172*; February 2000. See: http://www.arl.army.mil/~wildman/PAPERS/tr2172.html



NODES **(or *vertices*): set V = { A, B, C, D, E, F }**

LINKS **(*edges*): set E = { (A,B), (A,C), (A,D), (C,E), C,F) }**

GRAPH**: collection of vertices and edges: G(V,E)**

**A *Tree* structure is a "connected" graph with no "cycles,"
i.e., every node has at least one link to another node
and only one path exists between any two nodes.**

Via a link, a node can be a *parent* or a *child* of another node.

**A node without a child is called a *terminal or leaf node*
(e.g., the nodes at the bottom of the tree: B, D, E, and F)**

**A node with children is an non-terminal or *internal node*
(e.g., A and C);**

The *root node* is a special internal node with no parent (e.g., A).

Organization Charts are Trees (w/ boxes instead of circles)
(Often the name of the tree is inherited from the name of the root node - e.g., A):

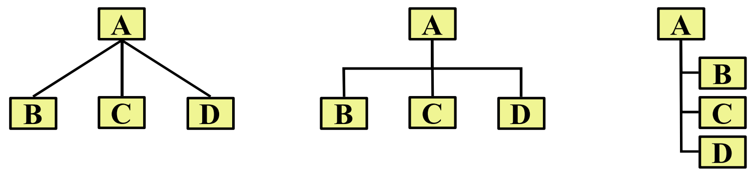


Figure 1: Tree Graph Definitions and Terms

There are many informal definitions of organization (or unit[8]). For example:

Webster's: *organization*—an administrative and functional structure[9], or

Joint Services Dictionary: *unit* (Department of Defense [DoD], North Atlantic Treaty Organization [NATO]) — 1. Any military element whose structure is prescribed by competent authority, … specifically, part of an organization[10],

Terms like "element" or "structure" are ambiguous in isolation; therefore, a more formal approach is required. Using the tree graph formalism, an organization can be defined as a node of a tree (e.g., node A in Figure 1), while the term "organization-association" can refer to a link of a tree (e.g., the line connecting nodes A and C, denoted by (A,C), in Figure 1). An organization chart is a graph composed of a set of nodes and a set of links, or in other words, a set of organizations and a set of organization-associations. In Figure 1, the set of organizations is {A, B, C, D, E} and the set of organization-association is { (A,B), (A,C), (A,D), (C,E), (C,F) }. Clearly, a set of organizations can be linked together in many ways, or in military terms, they can be task organized.

[8] Informally, the terms organization and unit are often synonymous.

[9] *Webster's 10th Collegiate Dictionary*

[10] See: http://www.dtic.mil/doctrine/jel/doddict

The goal of the force developer is to provide a common set of default organizations and organization-associations from which *any* other task-organized structure can be easily constructed. This default structure can be named the "default operational force structure" (DOFS) and can be maintained in concert with TOE and MTOE[11] data by the force development community in a database called an Army Organization Server (AOS). The AOS would be the authoritative source for force structure data and must support downloading of force structure data by tactical and administrative users across the joint services, and when authorized, by our coalition partners.

There are two key characteristics of a DOFS that greatly enhance military capabilities. First, it is high resolution and extends down to the *billet* level. In general terms, a billet organization is no different from a battalion organization; they are simple nodes of a tree graph that describe the aggregation and composition of clusters of people, equipment, and other organizations. A billet is the special case where the "cluster" is a single person. This means that billets are at the bottom of the organization tree (also called a leaf or terminal node). They may not have sub-organizations, that is, they are not "composed–of" any other organizations, so they identify the beginning of the aggregation process. Other than that, they are treated the same as any other organization. Billet resolution allows one to task organize (if and when required) down to the billet level. It also provides an ideal point to interface personnel and force structure data because they exist together in a one-to-one relationship.

The second characteristic of a DOFS is that it contains the default *operational*, not administrative, structure (unless, of course, the administrative structure is the operational structure). This means that all the small elements required to fight or deploy are included in the structure. This includes any special sections, elements, squads, teams, and crews[12]. For example, a tank crew is composed of four billet organizations: a tank commander, a gunner, a loader, and a driver. A basic rule is that platforms never fight alone; so two tank crews always work together to make a tank section. These organizations do not appear in current TOEs or MTOEs, one has to reference Field Manuals (FM) to discover this operational structure. Having the operational organization explicitly identified in the DOFS means that these organizations exist officially and are available for task organizing. If this is done properly, it should be a rare event to have to create new organizations while task organizing.[13]

To simplify the construction and dissemination of force structure data, there must be a simple way to uniquely identify the organizations (nodes) of the tree. This was the impetus for the organization identifier, or Org-ID, initiative. Org-IDs are to be enterprise-wide, unique values. They are *surrogate keys* meaning that they have no inherent meaning; or in other words, no information can be inferred about the data they identify from the key.[14] A surrogate identifier is

[11] TOE – Table of Organization and Equipment, MTOE – Modified TOE; the basic Army force structure documents.

[12] Conceptually, crews can also be considered equivalent to platforms or systems.

[13] For more information on DOFS, see ARL-TR-2172 (footnote 7) where the DOFS was previous called the default operational organization (DOO).

[14] Technically, a surrogate key is a primary key that has no inherent meaning, is composed of only a single attribute (database column), and whose value is not derived from any other entity. See: Lonigro, Mike, The Case for the Surrogate Key; http://www.dbpd.com/vault/9805xtra.htm.

not necessarily meant for human use. But users never need to know that they exist. However, the database management system (DBMS) and application programmers can use them extensively to greatly enhance performance and flexibility.

Originally, it was conceived that the force development community would assign each organization a unique 32-bit value, called an organization identifier, or Org-ID, that would stay associated with the organization for its life. Further, when created, every organization would have a default parent organization explicitly identified. This means that every organization (except the root node) would have a default position within a large Army DOFS. Thus, one could begin at any billet and iteratively follow the links to the parent organizations in a default path to the root node, for example, the Department of the Army node.

This concept naturally evolved for all data, and the specific notion of Org-IDs evolved into the general notion of EIDs for all data. With this background in mind, the following discussion describes major changes to the EID implementation approach as a result of this study.

3. Lessons Learned Since The Logistics Study

3.1 Decoupling Enterprise Identifiers (EIDs) From Organization Identifiers (Org-IDs)

Pursuant to an Army study on the applicability of EIDs to logistics[15], a simple but significant modification was made to the recommended implementation approach for allocating EIDs.[16] The original concept of EIDs arose from the operational realities of how the armed forces manage their resources, namely, through a force structure centric perspective. This is evident from the numerous data models and simulation models associated with battle command systems and studies. In other words, most of the data collected, maintained and produced by operational military units use force structure as a central concept to produce a unified view of the enterprise. Ultimately, somehow everything is related to force structure, whether one is building an operational plan (OPLAN), performing targeting tasks, planning re-supply operations, or executing installation management. The basic questions of who reports to whom, who owns what, and who is responsible for what permeates the enterprise information and its associations.

A popular set of models[17] is based upon the original Generic Hub data model[18]. These models have a core set of five battlefield entities: ORGANIZATION, PERSON, MATERIEL, FACILITY, and

[15] See Footnote 1.

[16] This was "Recommendation 8: The Materiel server should utilize an enterprise key scheme based upon DOOs."

[17] These models include:
C2 Core DM: *Command & Control (C2) Core Data Model;*
see: http://www-datadmn.itsi.disa.mil/ddm.html
CADM: *C4ISR Architecture Data Model*;
see http://www.c3i.osd.mil/org/cio/i3/AWG_Digital_Library/index.htm
AICDM: *Army Integrated Core Data Model*;
LC2IEDM: *Land C2 Information Exchange Data Model*, Edition 2.0, ADatP-32, 31 Mar 2000, and
ARCADM: Army CADM
see: https://arch-odisc4.army.mil/admg/html/datamodels.asp (CADM also accessible via this site)
JCDBDM: See Footnote 6.

FEATURE; ORGANIZATION is the basic entity for force structure. Data pertaining to it and all other battlefield objects are naturally associated, but it is the ORGANIZATION entity, be it a military combat unit or an administrative headquarters that forms the basic structure to which all other entities are ultimately associated.

This perspective induced the association of enterprise identifiers with organization identifiers. *The original approach built an EID from an Org-ID.* The idea was simple; an organization would receive an Org-ID from its force developer. Then, the organization could build EIDs using its Org-ID as a **prefix**. This process is illustrated in Figure 2.

An advantage of this approach is that an organization can use the Org-IDs under its control to tag all other battlefield data items under their purview. Because no two organizations participating in the scheme would ever receive the same Org-ID, the Org-ID could be used as a prefix to generate many other unique values. Consequently, EIDs were defined as a sequence of 64 bits created by concatenating a centrally apportioned, 32-bit Org-ID with a second 32 bit sequence generated locally. The resulting 64 bit EIDs would be guaranteed to be unique across the enterprise. This is illustrated in Figure 3.

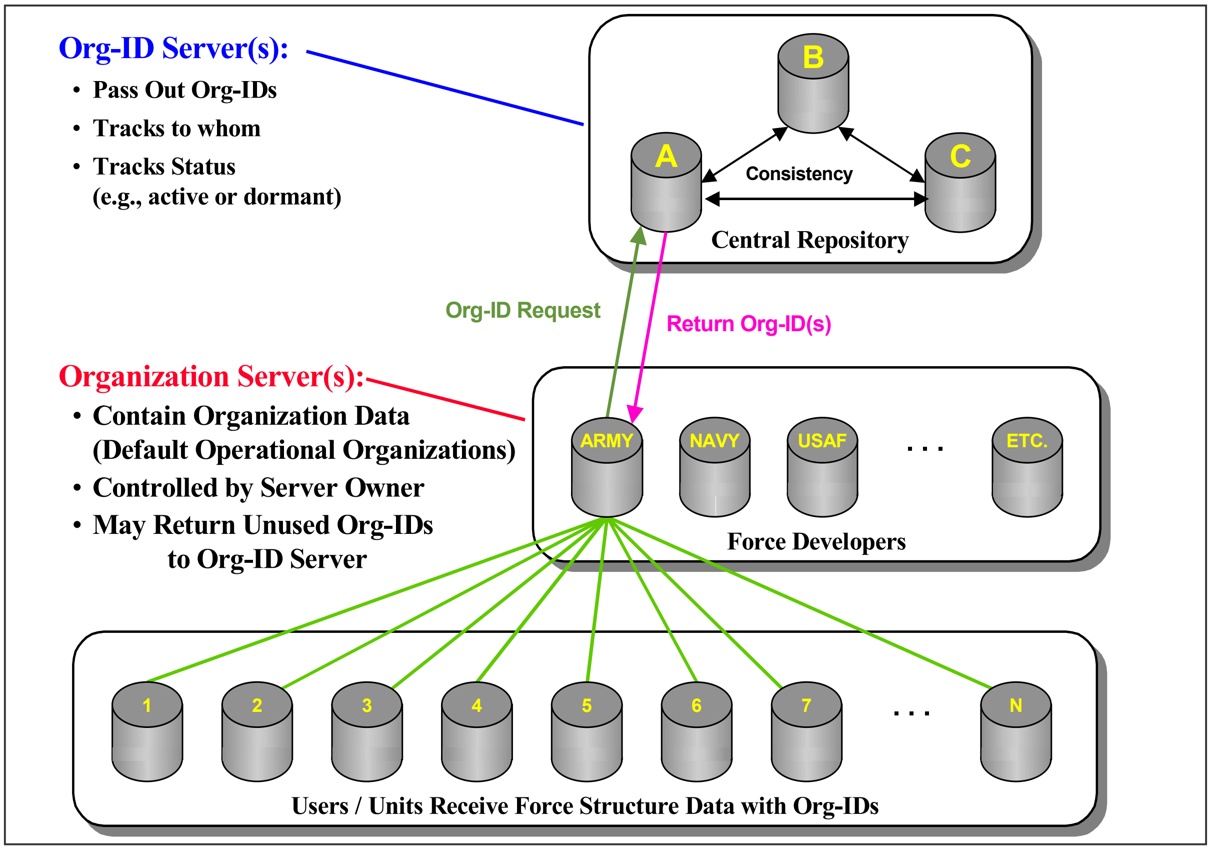
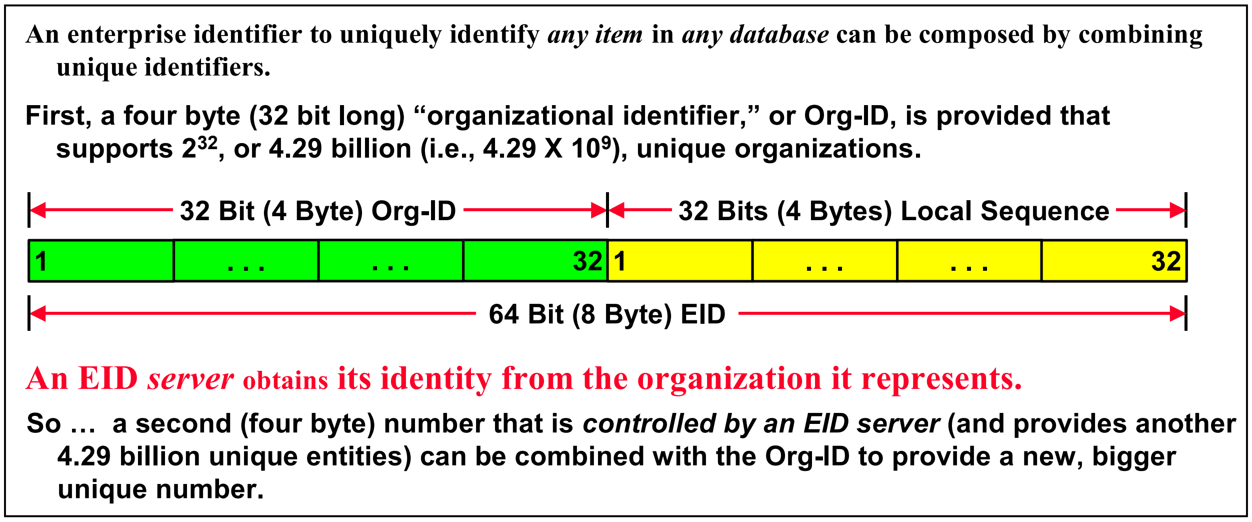

Org-ID Server(s):
• Pass Out Org-IDs
• Tracks to whom
• Tracks Status (e.g., active or dormant)
B
A
C
Consistency
Central Repository
Org-ID Request
Return Org-ID(s)
Organization Server(s):
• Contain Organization Data (Default Operational Organizations)
• Controlled by Server Owner
• May Return Unused Org-IDs to Org-ID Server
ARMY
NAVY
USAF
. . .
ETC.
Force Developers
1
2
3
4
5
6
7
. . .
N
Users / Units Receive Force Structure Data with Org-IDs

Figure 2: Org-ID Servers Provide ORG-IDs For Organization Servers That Maintain Force Structure Data

[18] Army Tactical Command & Control Information System (ATCCIS) Generic Hub 4 data model, ADatP-5 Draft 1.0, 1 Oct 1999; The predecessor of the LC2IEDM. See http://www.euronet.nl/users/atccis/index.html for historical information.



An enterprise identifier to uniquely identify *any item* in *any database* can be composed by combining unique identifiers.

First, a four byte (32 bit long) "organizational identifier," or Org-ID, is provided that supports 2^{32}, or 4.29 billion (i.e., 4.29 X 10^9), unique organizations.

An EID *server* obtains its identity from the organization it represents.

So … a second (four byte) number that is *controlled by an EID server* (and provides another 4.29 billion unique entities) can be combined with the Org-ID to provide a new, bigger unique number.

Figure 3: Original Scheme to Create EIDs from Org-IDs

Pursuant to the logistics study, it became apparent that the requirement to first allocate Org-IDs to military units to implement EIDs added unnecessary complexity and delay. Further, it limited the number of EIDs per organization to a fixed number (4.3 billion). This may be too huge for many organizations, and too small for others. The original process requires that an organization receive an Org-ID from the force development community before it can create data using EIDs. As MTOE/TOEs get updated annually, Org-IDs may change further complicating the process. Although it is expected that battle command processes and data models will continue to be organization-centric, for simplicity and functionality reasons, it is advantageous to develop a ubiquitous battlefield object identification system that is independent of the force development process. In other words, there are no technical reasons why globally unique, 64-bit identifiers (i.e., an EID) should be dependent on the force structure process.

3.1.1 EID Server Architectures

Separating Org-IDs from the EID generation process greatly simplifies the task of EID distribution. There are numerous techniques that can be employed to allocate (32) bit sequences that are guaranteed not to have been distributed to anyone else. The study team has explored various implementation strategies based on the concept of having one or more EID "seed servers" (ESS). After subscribing to the ESS, members can receive EID prefixes and then proceed to build EID servers (ES) to locally generate 64-bit sequences for tagging records, objects, and other data. Figure 4 illustrates an architecture to accomplish this task.[19]

The architecture includes three strata: EID users (on the right), EID servers (in the middle), and EID seed servers (on the left). Any time data is created, it is uniquely tagged with an EID. This is accomplished by obtaining an EID from an EID Server (ES). An ES is *any* computer program that provides EIDs to requestors. As illustrated in Figure 4 and Figure 5, an ES creates 64-bit

[19] S. Chamberlain, "An Enterprise Identifier Strategy for Global Naming Across Arbitrary C4I Systems," Proceedings of the 6th International Command & Control Research & Technology Symposium, USNA, Annapolis, MD, 19-21 Jun 2001; Presented 19 June. See: http://www.arl.army.mil/~wildman/PAPERS/6thc2rt.html, or http://www.dodccrp.org/6thICCRTS/Cd/Tracks/Papers/Track2/059_tr2.pdf.

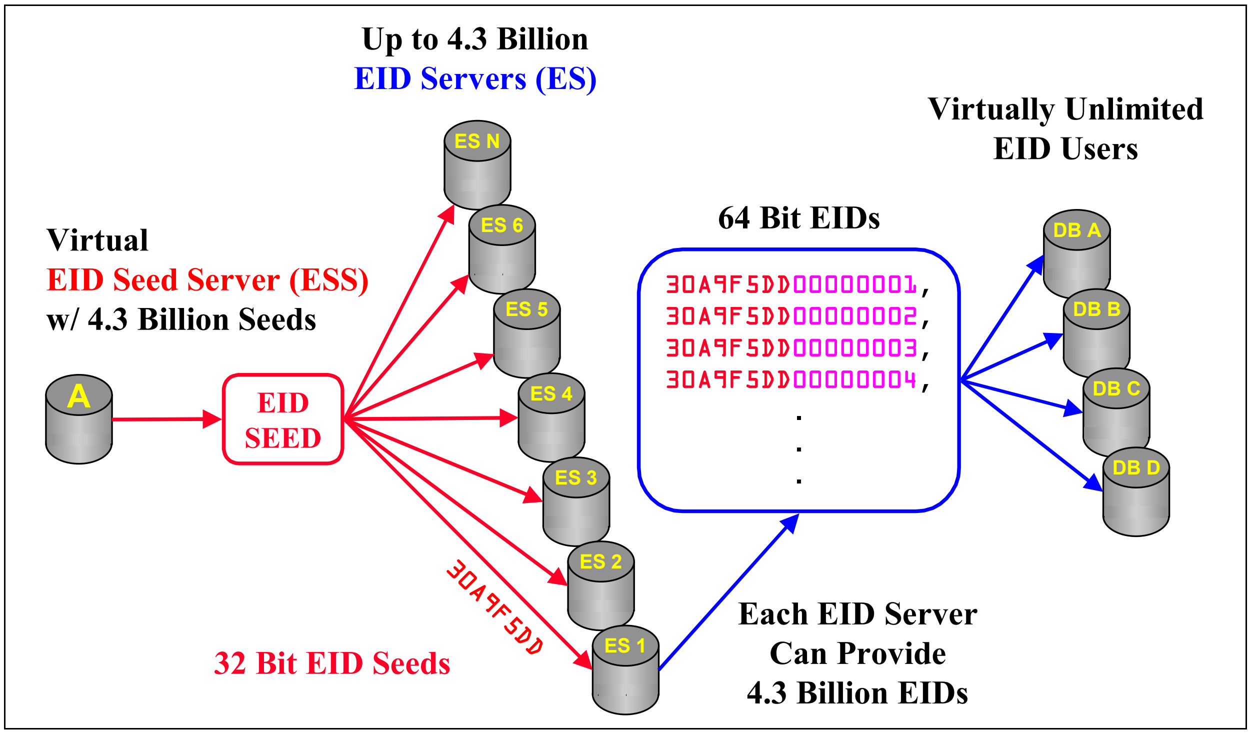


Figure 4: EID Server Architecture

EIDs by appending a locally generated 32-bit EID suffix to a 32-bit EID prefix. The EID prefix is an EID seed that is obtained from an *EID seed server* (ESS). In Figure 4, the EID values are represented using hexadecimal notation where each character (with a value of '0'–'9' or 'A'–'F') represents a sequence of four bits. Therefore, 16 characters are used to denote a 64-bit EID.[20]

An ESS is a member of a special set of redundant servers that pass out EID seeds to registered users. Since an EID seed is a 32-bit value, 4.3 billion ESs may be established.[21] Because an EID suffix is also a 32-bit value, each ES may generate 4.3 billion EIDs suffixes for its EID seed. In

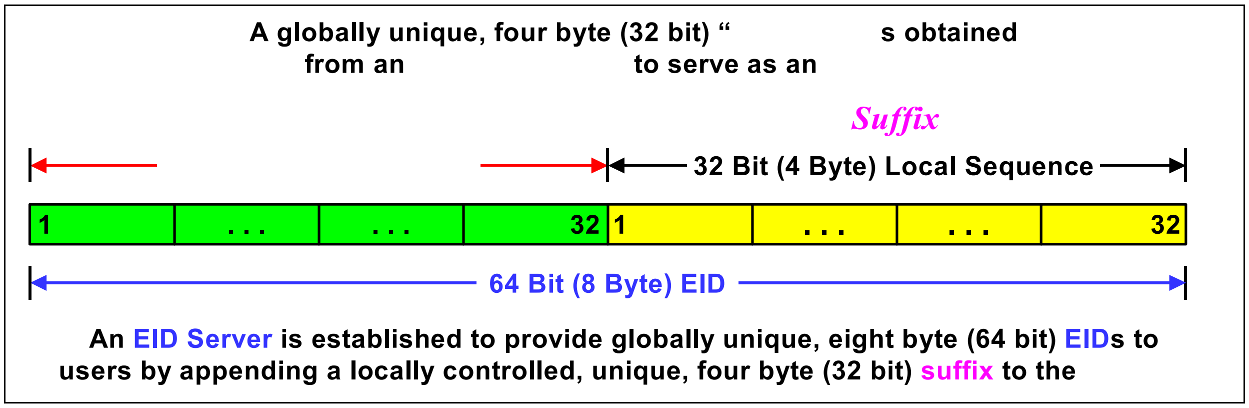

A globally unique, four byte (32 bit) " s obtained
from an to serve as an
Suffix
32 Bit (4 Byte) Local Sequence
1
. . .
. . .
32
1
. . .
. . .
32
64 Bit (8 Byte) EID
An EID Server is established to provide globally unique, eight byte (64 bit) EIDs to
users by appending a locally controlled, unique, four byte (32 bit) suffix to the

Figure 5: Structurally, EIDs Based Upon EID Seeds Are Identical To Those Based Upon Org-IDs

[20] Note: each of the 16 characters ('0' – '9' and 'A' – 'F') denotes one of the 16 combinations of four bits 0000 through 1111. The fact that 16 characters are also required to denote a 64-bit EID is a coincidence.

[21] The actual value is 2^{32}, which is 4.295 billion.

other words, this technique supports 4.3 billion ESs, each capable of providing 4.3 billion EIDs, for a total of 18 billion-billion EIDs.

There is always a debate about address space size when this approach is used. The goal of this implementation is to select the smallest possible, common size that is capable of accomplishing the task. This choice is driven by bandwidth constraints, such as exists over tactical, wireless, communication systems. Many of the benefits of EIDs are accentuated within these environments. If necessary, the length of an EID can be *increased* in the future without significant difficulty.[22]

3.1.2 EID Seed Servers (ESS)

ESSs are a special set of controlled servers that pass out 32-bit EID seeds to registered users. They form a virtual server because, externally, they appear to be a single resource although several tightly coupled, redundant machines would be used to provide robustness, rapid speed of service, and backup.[23] To establish an ES, one must obtain an EID seed from an ESS, which requires obtaining an ESS user account. Obtaining a user account is not a difficult procedure, but it is required. Anyone may request an account and they are currently granted freely to encourage EID usage and to gain an experience base. Ultimately, accounts may be scrutinized and limited to agencies and individuals with a bona fide requirement or official capacity to create official enterprise data. Government examples are officials in corporate information offices (CIO), acquisition organizations, such as program executive officers (PEO), project managers (PM), and systems builders, and the simulation community. Currently, human intervention is included as part of the authorization process. Eventually, it is anticipated that this will only be required for non-traditional cases.

A prototype ESS is operational at **https://ess.arl.army.mil**. Registration occurs via a browser interface over a secure connection. To register, a person must provide reliable point of contact (POC) information and the position of the POC (e.g., Assistant PM for the XYZ systems) or the reason for the account. As with many web-based systems, a valid email address is a key account requirement. Once an ESS user account is established, a user may request EID seed accounts. Although one EID seed account is normal, several accounts may be established. Each EID seed account is for a specified usage level that designates a number of seeds. The usage levels are: normal (one seed), moderate (10 seeds), heavy (100 seeds), and special (over 100 seeds). The usage level can always be increased at a latter time. Recall that one EID seed can be used to produce 4.3 billion EIDs, so large blocks of EID seeds would only be required for cases of highly distributed and autonomous systems (e.g., warrior and on-board command and control and weapon systems).[24] Once an EID seed account is obtained, the account owner can obtain EID seeds and establish ESs to provide EIDs directly to data creators.

[22] Database fields can be increased in length without losing data. If a size change is affected before the top bit of the EID Seed is used (i.e., before the 2.1 billion, or 2^{31}, value is reached) then it can be used as a discriminator if both 64-bit and a larger sized EIDs coexists.

[23] Currently, the ESS is a single machine. In the future, it will be distributed among several machines.

[24] For example, a PM may be deploying thousands of systems that each require an autonomous data creation capability. This can be accomplished in several ways, to include providing individual EID seeds to each system, or by dividing up the 4.3 billion EIDs producible from a single EID seed.

An ESS can provide other services in addition to allocating EID seeds. One of these is a EID tracking service. Recall that a primary advantage of EIDs is that they facilitate the creation of arbitrary associations between disparate data located within independent databases. In this environment, it is inevitable that one will occasionally receive an EID that references data unknown to the local system.[25] For these cases, the ESS provides a tracking service via the "Find-EID" command. A user simply enters the unknown EID or EID prefix and the POC information about the user that owns the EID seed is provided.

However, this is only the first step of the process. When establishing an EID seed account, a user may enter a URL[26] of a host that will provide a tracking service for the EIDs produced from the EID seeds received under that account. This is completely at the user's discretion.[27] The tracking service can be on any host selected by the user — it does not have to reside on the ES established for the EID seed. This is to allow proxies to be used in situations where users do not want to advertise their ES(s) due to security or other reasons. If a tracking service URL is provided by the EID seed account holder, then the Find-EID command is forwarded to that host and the EID trace continues until the data repository containing the item tagged by the EID is reached. At the repository, the Find-EID command is converted to a "Fetch-EID" command that returns XML[28] code that describes the structure and values for the unknown data tagged by the requested EID. This code is returned to the user that initiated the Find-EID request. It is emphasized that an ESS contains no information about the data tagged with EIDs, but only the POC information provided by the ESS account subscriber, which may include a tracking service locator URL.

The tracking service can also be used as a validation service. An ES can contact the ESS to verify that it is using a correct EID seed. Because humans receive EID seeds, it is plausible that an EID seed can be entered incorrectly into an ES when it is established (or is being re-initialized after a malfunction). An ES can contact the ESS and use the Find-EID command to verify that it is using a valid seed. By entering its own EID seed as the prefix, the resulting POC data can be compared to the expected values.

3.1.3 EID Servers

Once an EID seed is obtained, an ES can be established to distribute its 4.3 billion EIDs. Recall that an ES is any program that provides EIDs to one or more users. Upon accepting an ESS account, a subscriber agrees to maintain ESs that adhere to four strict constraints:

(1) all EIDs distributed by ES programs must be a 64 bit sequence composed of a bona fide, 32-bit EID seed prefix allocated to the subscriber;

[25] This should be a rare event in a tactical system where rigorous control is maintained over the data.

[26] URL: Uniform Resource Locator – a web address.

[27] This is analogous to Internet domain name service requests. Not all IP addresses are registered. This is the prerogative of the IP address owner.

[28] XML – The Extensible Mark-up Language, a standard for describing the structure of information. Proposed standard ways for capturing the structure of the data are being developed based on XML, such as XMI (XML Metadata Interchange). In addition, large software producers are adopting XML as a data transfer mechanism. See http://www.w3.org/XML/.

(2) EIDs are maintained in either binary form as a single 64-bit field, or two 32-bit fields, or as 16 characters using hexadecimal notation,

(3) the ES program must ensure that the EIDs it produces are never duplicated. In other words, it must ensure that its locally generated 32 bit suffixes are always unique (i.e., it never allocates the same suffix twice), which implies that the ES must have some type of backup scheme to prevent re-use in the event of a power loss or major malfunction.

(4) the ESS subscriber's POC information must be kept current.

Other than these four constraints, the ES owner has autonomy in how the ES is implemented.

Recall that an ES can be as simple or complex as its designer requires. The ES owner decides most implementation details, such as who is allowed to access the ES, how access is protected, and how EIDs are allocated. Access to an ES can be categorized as *single-user* or *multi-user*. A single user ES limits access to a single program, machine, or user. This may be the preferred approach for isolated systems, such as those with individual warriors (e.g., forward observers or special operations teams) or for personal digital assistants (PDA).[29] A multi-user ES allows a variety of users and may be *open*, meaning anyone may obtain EIDs from it, or *restricted*, meaning that one requires permission to obtain EIDs. The ES owner decides whether protections, such as passwords or link encryption, are required

The ES owner specifies how EIDs are allocated. The scheme does not need to be sophisticated. For example, a simple technique is to merely add one to the previous EID suffix for every EID request (i.e., 1, 2, 3, … , up to 4.3 billion). Such a mechanism would require just a few lines of Java® code to implement a single-user, embedded ES (e.g., like that used in a PDA). At the other extreme, a large, multi-user ES may use more sophisticated EID allocation schemes. For example, the EID suffix space may be divided into segments so that each system has it own block of EID values. Regardless of the allocation scheme used, to meet the uniqueness criteria, some type of persistent backup is required. This is to ensure that duplicate EIDs are never produced as a result of an ES malfunction or system fault. The details of how this is accomplished are a local implementation decision.

Like the ESS, an ES may offer a tracking service. To do this, the ES must maintain a record of an EIDs destination. For a single-user ES this is a trivial task since it is providing EIDs to only one destination. For a multi-user ES, this requires additional functionality akin to the ESS.[30] In any case, the ES owner always retains control over its operation. For example, it is perfectly permissible to limit responses to tracking service requests based on any criteria deemed appropriate by the ES manager.

3.1.4 EID Server Location Flexibility

The new way in which EID prefixes are obtained does not result in a change from the original concept to the deployment options for ESs. This section is a review of those options from the Logistics Study (for completeness), since they are applicable to integrating administrative and battlefield automated personnel systems.

[29] For example, a Palm Pilot.®

[30] At this point, one should recognize the striking similarity between an ESS and a multi-user ES. They are essentially equivalent with the exception of the size of the bit sequence they provide (i.e., 32 versus 64 bits).

The elegance of this architecture is its flexibility. When an AIS creates data, it must obtain an EID from an ES. The AIS and ES do not have to be co-located and their proximity will depend upon policy, performance, and security issues. An ES may be embedded within an AIS, it may be located onboard a common platform (e.g., computer), or it may be accessed via a local or wide area network (LAN, or WAN, respectively). These cases are illustrated in Figure 6.

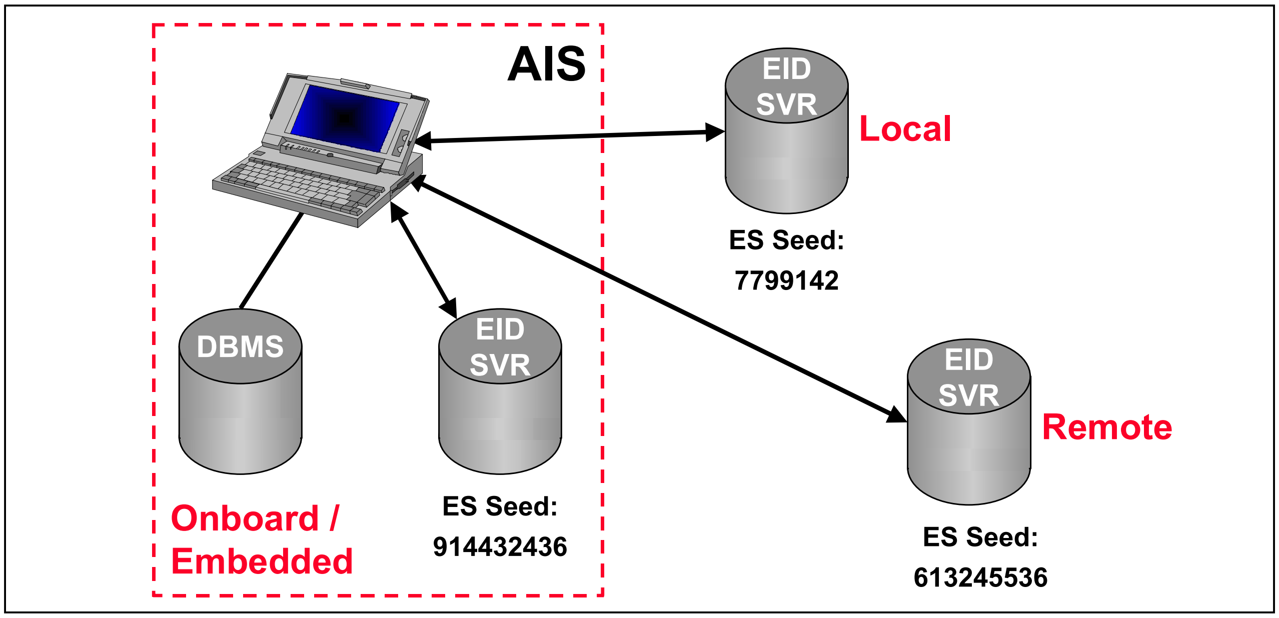


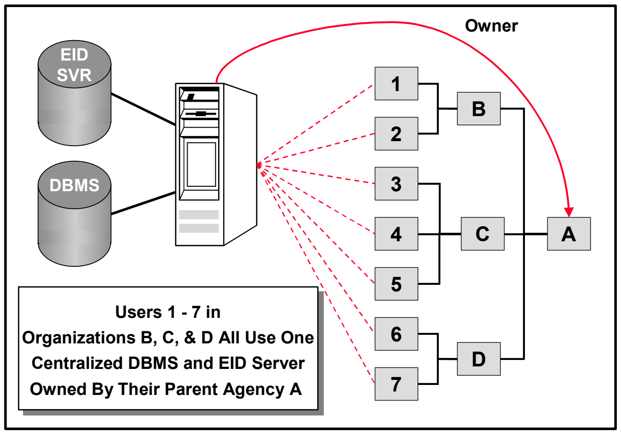
Figure 6: EID Server Location Options

Usually, performance will be the driving force behind proximity decisions for ESs and AISs. Certainly, isolated computer systems, like those carried by soldiers or on combat vehicles, will have onboard ESs tightly integrated with the database management system. Command posts and tactical operation centers could choose onboard or redundant ESs accessible on a LAN. Highly controlled, tightly coupled applications may choose to access a common ES via a secure WAN. Configuration control is immensely flexible and the options are boundless. The design decision belongs to the system managers.

To demonstrate these options, consider four extreme cases:

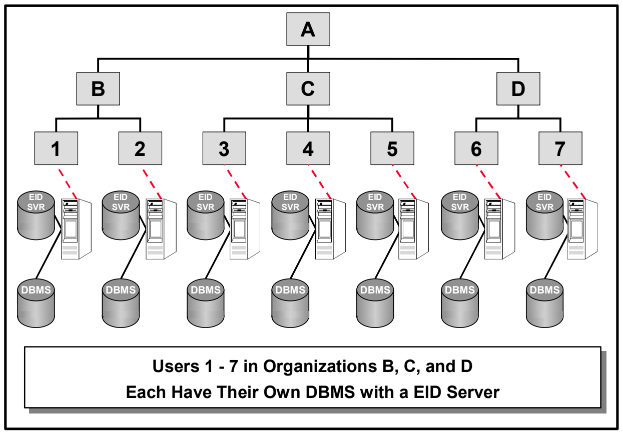
- centralized database with collocated ES,
- decentralized databases with collocated ESs,
- centralized database with decentralized ESs, and
- decentralized databases with a centralized ES.

The first two examples illustrate cases in which an AIS is collocated with the ES it uses. Figure 7 illustrates a completely centralized situation in which every user utilizes programs on a centralized machine, perhaps via a browser, to create and manipulate data. At the other extreme, there could be 4.3 billion independent, decentralized AISs each with their own embedded (or collocated) ES. This is the completely decentralized case, illustrated in Figure 8 that is exemplified by the extreme case of 4.3 billion users each with a wearable computer. Because each AIS has its own ES each using a unique EID seed, all the data created by the 4.3 billion users is guaranteed to be tagged uniquely (i.e., there will be no collisions).



Owner
EID SVR
DBMS
1
2
3
4
5
6
7
B
C
D
A
Users 1 - 7 in Organizations B, C, & D All Use One Centralized DBMS and EID Server Owned By Their Parent Agency A

Figure 7: Extreme Case – Centralized



A
B
C
D
1
2
3
4
5
6
7
EID SVR
DBMS
Users 1 - 7 in Organizations B, C, and D Each Have Their Own DBMS with a EID Server

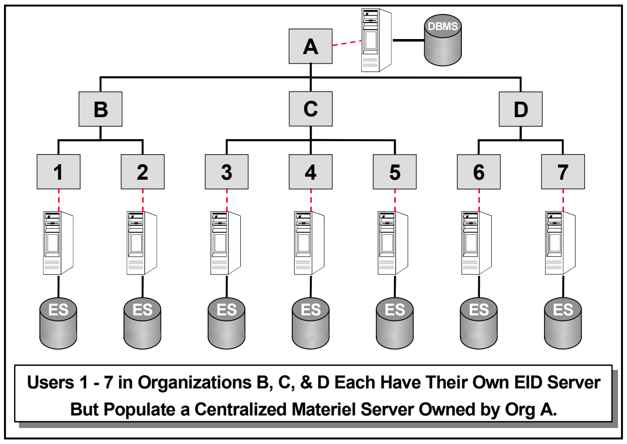
Figure 8: Extreme Case – Decentralized

There can also be a mixture of these two extremes. If a high-speed network is available, an AIS does not have to be collocated with the ES it uses. Figure 9 illustrates a centralized AIS that has many, distributed ESs located with the system users. As each user inserts data into the centralized AIS they provide their own EIDs to uniquely tag the data. For example, each user could have its ES embedded in a smart card that can be inserted into a local machine to create data in the centralized database under its authority.

Conversely, Figure 10 illustrates a set of distributed AISs that share a common ES. To insert data into their local AIS, the user must obtain an EID from the central ES. The central ES may also include functions to check that other constraints are satisfied. For example, this might be a case where people are responsible for a portion of a distributed database and the central ES can check that they create only the data that they are authorized. These four examples show the extreme cases. Real configurations will reside somewhere in the middle of these extremes. The advantage portrayed by this ES approach is that each system builder may implement their ES configuration based upon their own local policy, procedures, and performance requirements.

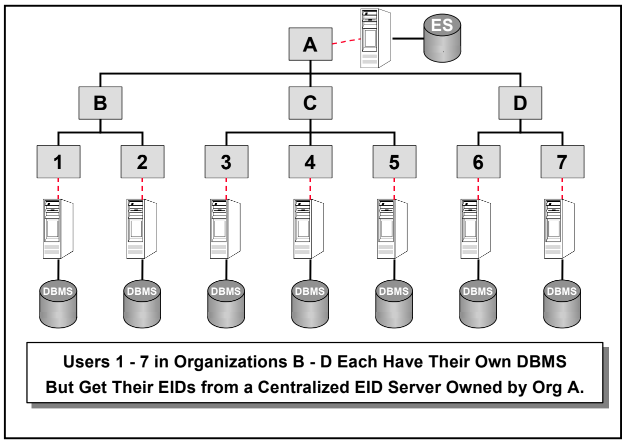
In summary, an EID server:

1. Must provide 64 bit EIDs (or its hexadecimal equivalent) using a bona fide EID Seed prefix.
2. Must never duplicate an EID. This implies that it must have some type of backup scheme to prevent re-use in case there is a loss of power or major malfunction.



A
DBMS
B
C
D
1
2
3
4
5
6
7
ES
Users 1 - 7 in Organizations B, C, & D Each Have Their Own EID Server But Populate a Centralized Materiel Server Owned by Org A.

Figure 9: Mixed Extreme One



A
ES
B
C
D
1
2
3
4
5
6
7
DBMS
Users 1 - 7 in Organizations B - D Each Have Their Own DBMS But Get Their EIDs from a Centralized EID Server Owned by Org A.

Figure 10: Mixed Extreme Two

3. Must be established by an authorized person who is registered with the ESS and maintains current point of contact (POC) information. The subscriber may change any POC data, but it must be reflected at the ESS. [Note that *the enterprise* is now defined as the set of users of the ESS and is not constrained by service, governmental, or national borders.]

Nearly all ES implementation decisions are made at the discretion of the ES owner. Features such as optional services (e.g., a tracking service), encrypted links, login passwords, and single versus multi-user functionality as well as other access controls and security constraints are all local decisions. The goal is to maximize flexibility via local control and minimize bureaucracy once an EID seed is obtained.

3.2 EIDs as Alternate Keys

Although there are significant advantages for database maintenance when using EIDs as primary keys, all the interoperability advantages are equally obtained when they are used as alternate keys. Thus, EIDs can be implemented without modifying the existing primary key system in a legacy database.[31] If the legacy system is a relational database, three tasks are required to implement EIDs. First, a single new column is added to each existing table to hold an EID. Second, an EID is assigned to each of the existing rows. Third, the insert routing is modified so that every subsequent insertion populates the EID column with a new EID. The existing applications are unaffected and the EIDs can be exploited if and when the system managers deem necessary.

To support future capabilities, the legacy system maintainers may implement two relatively simply features. First, a "Fetch-EID" command, as was previously described, is required if one wants to support the EID tracking function. This is a command that, given an EID, finds the row of the database that is tagged with the EID and returns XML code that describes the attributes of the row. To facilitate this feature, it is helpful to add and maintain a simple table that maps each EID to the name of the table in which it resides. This allows one to rapidly identify the table in which to search for the EID.

3.3 Application Of The EID Concept To Reference Libraries

One of the topics of the logistics study included organizing large sets of materiel data into what is essentially a reference library. There are analogous topics in the personnel community. One way to categorize data is by its perishability. Using this perspective, data may be categorized as static, stationary, and dynamic. *Reference data* is static data. Examples are lookup table entries for state, airport, or country codes. At the other extreme is *dynamic data*. As its name implies, it is constantly changing and must be updated frequently to remain synchronized. Situational awareness (SA) data is in this category. The third category lies between these two extremes. *Stationary data* is information that is "semi-static," meaning that it is relatively invariant over its lifetime and its periodicity is known and of significant duration to allow it to be treated as static

[31] See Annex A at the end of the main document for a detailed analysis of implementation issues and adoption of EIDs in legacy systems as well as planned systems.

data; that is, it may be reasonably maintained in a shared reference library.[32] An example of stationary data is phone numbers. Although thousands of phone numbers are added, deleted, and changed daily worldwide, it is not a frequent event for a person's phone number to change. Usually, a person's phone number is static for the duration of their tour of duty. Therefore, phone numbers can be reasonably maintained in a reference library, called a phonebook, that is only distributed once per year. The small subset of new, changed, and deleted phone numbers is handled via other means (i.e., directory assistance). Consequently, although stationary data is not static, it is "static enough" to treat it as reference data (i.e., maintained in a reference library).

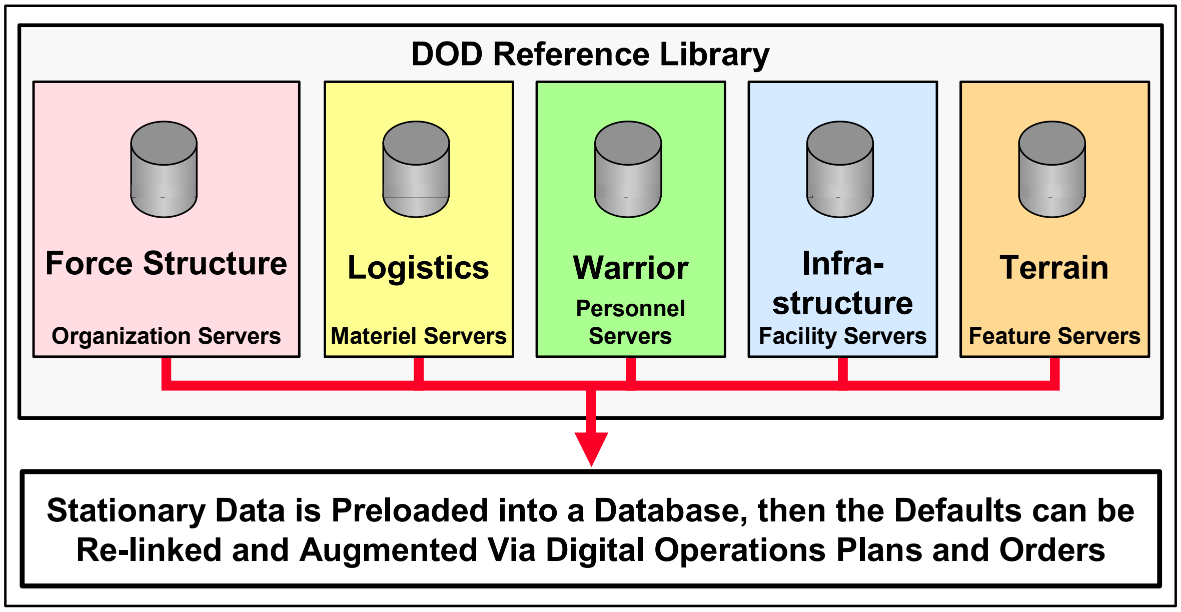


Figure 11: Suite of Servers that Contain Stationary Data

A large subset of personnel data can be categorized as stationary data. Information such as codes for military occupational specialties (MOS), additional skill identifiers (ASI) and skill qualification identifiers (SQI), and skill levels are stationary data. Just as with materiel data, it makes sense that this information should be defined and maintained by the specialists who do it for a living. In other words, battle command systems users should obtain this information on a periodic basis from an authoritative source.

Recall that several C4ISR[33] oriented data models include a common set of five basic battlefield entities: ORGANIZATION, MATERIEL, PERSON, FACILITY, and FEATURE. These five data domains contain large amounts of information that can be categorized as stationary data. This means that the data can reside in common reference libraries, provided via data servers, and can be periodically downloaded into operational battle command systems; see Figure 11. Three of these domains, organization, materiel, and personnel, will provide the initial focus of an Army Organization Server (AOS). By rigorously controlling the update process of the stationary data,

[32] Originally, stationary data was called reference data and reference data was termed lookup data. However, when it was learned that a US Transportation Command initiative was in progress to standardized lookup tables, and they called it reference data, the names were modified to be aligned with their study and avoid confusion.

[33] C4ISR: Command, Control, Communications, and Computers, Intelligence, Reconnaissance, and Surveillance.

users can be confident that they have a consistent set of reference material preloaded into their computers.

EIDs can be used to provide a common naming scheme across the reference libraries (i.e., for both the reference and stationary data) so that the common data will include a common set of EIDs. Once users have downloaded a common set of reference libraries, they may refer to the ITEMS by passing the terse EIDs instead of the bulky data, thus significantly reducing the bandwidth required to manage this subset of C4ISR information.

There are many advantages to using meaningless values to identify data. One of the principles for using EID is that two items with the same EID must be semantically equivalent. In other words, one can locally add or modify the attributes of a data item from another source provided the meaning of the data item is never changed. An example is that the name of a unit can be changed from English to German and still retain the same EID because the EID still references the same unit.

Consider static reference data, for example, country codes. Suppose that an official table exists that contains the official country codes for the world, one record for each country. Further, suppose that each country gets to pick what it wants to use for a two-character country code (a business key). Hidden from the user is the fact that each record is tagged with an enterprise-wide, unique EID. When an association with a record is required, or when machines exchange data, it is the EID that is of primary interest, not the two-character code. Although the United States may pick a two-character code of “US” for its official entry, other values can be attached to the same EID provided they “mean” the same thing. For example, different languages will be required, so an alias for “US” might be “EU” (in a Spanish system), or “VS” (in German), or “SU” (in Italian). As long as it is attached to the same EID that resides in the official table (the authoritative source), it will reference the same semantic entry (i.e., the country known by the name of the “United States of America.”).

Because reference table codes usually have meaning to them (i.e., “US” for United States), they are highly influenced by change. However, since an EID has no information encoded into it, there is never a reason to change it. This helps to disambiguate the maintenance of standard codes. Clearly, the same code value (e.g., “US”) can exist in many tables, so a code without a table name is meaningless. However, as codes get updated, their EID does not change unless the semantics of the code also change (e.g., Czechoslovakia [CZ] became two new countries, the Czech Republic [EZ] and the Slovak Republic [LO]). As enumeration becomes popular (i.e., numbering the table values from 1 to N), EIDs become even more useful since tables will contains many of the same value. EID are very useful because they unambiguously track the evolution of the meaning of a code. An EID denotes one, and only one, entry in one table in the entire enterprise. Thus, if a code changes but the meaning does not, the EID can remain. If the meaning changes, an EID can be clearly linked to its successor or successors. This general technique can be applied to any reference data (e.g., blood-type code) or stationary data (e.g., the title of an MOS).

3.4 Data Ownership, Authoritative Sources, and Duplication

To provide a consistent product in an environment where the lines of ownership for reference and stationary data are rarely perfectly clear, it is important that authoritative sources be identified, sanctioned, and funded. For personnel data, like materiel data, it is difficult to identify a single, definitive creator for all domain data. Therefore, identifying a responsible data source (i.e., maintainer) may be the best option and this is often policy driven rather than technology driven. Using EIDs makes the task of assigning a unique identifier to a new soldier technically easy. However, deciding who is authorized to execute this task is a policy decision. When using EIDs, it is technically viable to allow the recruiter to create the initial record about a soldier and thus assign the permanent EID to be used by all other systems. This is an example of the decentralized control scheme illustrated in Figure 8.

EIDs do not solve the duplication issue.[34] They are assigned when data is created and there is no technical way to prevent two different users from creating duplicate entries in their respective MATERIEL tables and tagging them with different EIDs. Preventing duplicates can only be accomplished via policy. For example, there is no way to prevent both the Army and the Navy from creating data about Army personnel and each assigning them different EIDs. The only solution is to state a policy that the Army is the authoritative source of information about US Army employees, regardless of who creates it. In this case, the Army (e.g., PERSCOM) would take responsibility for collecting and maintaining the authoritative source of its civilian employee data. This doesn't mean that others cannot create Army personnel data, only that the Army controls who is allowed to create the data. The point being that an EID system cannot prevent unauthorized duplicates from being created, but it can ensure that only one authoritative source is identified and sanctioned.

A good example of a strategy to implement such a policy is illustrated in Figure 10. Consider a system such as DIHRMS.[35] The assignment of EIDs could be controlled at the DoD level, yet each service would be responsible for entering its own employees. In Figure 10, node A is the DoD, and nodes B-D could be the Army, Navy, and Air Force. How each service obtains its EIDs is an implementation issue. Blocks of EIDs could be assigned a priori, or they could be obtained in real time provided sufficient bandwidth was available between the databases and the ES. The same approach could be used by each service. EIDs could be assigned in some central personnel facility, or it could be accomplished at the recruiter level when they enter the data for a new recruit. Any configuration is possible, but in each case some authority must designate an authoritative source and define the control policy.

Clearly, a significant benefit is achieved when the chosen authoritative source is an agency already responsible for the data (i.e., the proponents). This helps ensure that the data is not only consistent, but timely and accurate as well. To accomplish this, a proponent for the data must be identified to rigorously control the creation, collection, maintenance, and deletion of the domain data. A reasonable practice is to begin with the agencies that already maintain the data as part of their charter. For example, every service has a force structure development community that handles a wide variety of force structure documents. These agencies should be the owners of the stationary and reference data for organizations. The same is true with materiel and personnel data.

[34] The implementations discussed here guarantee so-called *one-way uniqueness* only. In other words, no two EIDs will be identical at the time of creation and assignment. Achieving *two-way uniqueness* within the enterprise is essentially a policy and procedures issue.

[35] See footnote 2, pg 6.

3.5 User Access To EID Servers

For an EID system to work, it must be easily accessible to users while still offering a sufficient degree of protection. This was a major impetus for the redesign of the EID allocation scheme away from one that was Org-ID-based. The technologies required to provide user access to the ESS have reached a sufficient level of maturity so that it can be fully Web-based, provide sufficient degree of security, and have adequate traceability.

The current design and policy allows *anyone* to request an ESS account. This has a significant ramification on the scope of one's "enterprise." Under this model, the *enterprise* is defined by the set of ESS account subscribers. There is no service, governmental, or international boundaries created with this approach. The set of users can include military services, other government organizations, private volunteer organizations, industry, academe, or any other organization regardless of country of origin. This provides complete support for joint, coalition, and peacekeeping operations. Such an enterprise highlights why it is important to have a simple, common format for identifiers. In this environment, SSNs are *not* universal.

There are two characteristics of EIDs that allows this openness and flexibility. First, an EID seed reveals nothing about the data it is used to tag. All that is known by the EID server is that someone obtained a set of EID seeds (one or more). How the EID seed is used is completely up to the owner. Just because data is tagged with an EID does not mean that it is shared – it simply makes it easy to reference and share if desired. Therefore, data owners retain complete sovereignty over their data and share only what they desire while still participating in the EID system. This facilitates ease of data sharing without requiring it. Second, an EID is *not* a security mechanism—it is only an identification mechanism. It is presumed that, like other critical data, data tagged with EIDs are protected from basic security risks[36] just like other data. EIDs do not alleviate the requirement for cryptographic and other means of protecting data.

To ensure easy access, a web-based access point is used. By dividing ESS access into two phases, registration and allocation, simplicity is maintained. Users register to get an ESS user account. This is where the bounds of the enterprise are determined. Those given accounts are part of the enterprise from an EID perspective. Recall that once a user account is obtained, subscribers may request allocations of EID seeds via EID seed accounts. This is a separate action and should not be a frequent activity. Recall that one EID seed can be used to create (i.e., tag) 4.3 billion data items. Once the seed is obtained, the user may proceed completely independently from the ESS.

3.6 Choice Of Optimal Taxonomies

During the logistics study, a plethora of taxonomies for materiel and logistic items were discovered. Although not as copious, several variations of perspective were also discovered for the personnel domain. As with any domain, there is no "correct" taxonomy, so the choice of how to categorize and compartment the data is "left to the implementer." A primary problem is one of overlapping semantics. Many different models refer to the same concept or object from different perspectives. What would be helpful is a simple, unambiguous way to cross reference

[36] For example, integrity, confidentiality, authenticity, and non-repudiation.

common entities, across a wide variety of models and taxonomies, without having to have pre-arranged data structure mappings. This is where EIDs are of great utility.

Recall that EIDs provide a way to conveniently reference any object, entity, or fact regardless of its data storage technology or structure. This allows one to build arbitrary relationships between items from disparate data sources. Because EIDs have a common format with no inherent intelligence, any object can be referenced without knowing its form. This allows a single reserved field[37] to refer to any object in the enterprise. Therefore, a single table of two attributes, both EIDs, can be used to map analogous items. This does not alleviate the problem of having to select data structures for the authoritative source, but it does provide the users of the source a convenient facility to build their own data clusters from those of the source.

Once an unambiguous reference is obtained, syntactic tools and standards, like those associated with XML, can be used to get detailed structural and composition information about the object tagged with the EID. This allows the information to be entered into an AIS. If every AIS provided a single external function called, say, *Fetch_EID*, that given an EID returned the XML code defining the entity, then one could always obtain the structural data for any EID tagged item regardless of its storage technology[38].

Although the capability of being able to share syntactically equivalent information between arbitrary data sources is the first enabler for interoperability, it does not solve the problem of semantic equivalence. Conversely, semantic equivalence does not imply syntactic equivalence. These are two different problems and EIDs focus on semantic equivalence. Depending upon the resolution of the semantic definition, one can overload the definition to allow direct correlation even though the syntactic definitions may vary. Consider a data item that denotes the location of a particular physical object. If the formats of the location attributes are not tightly specified, then two data entities can share a common EID. For example, one location for object X could be represented in latitude-longitude and another in UTM coordinates. Both data items are semantically equal; they represent *the location of object X.* They just happen to be different syntactically. This difference could be different languages, units, or representations. But they denote the same thing. If the semantic definitions were changed to *the location of object X in UTM Coordinates*, a common EID could not be used for both items.[39]

The specificity of the semantic definition determines the amount of overlap permitted. When combined with syntactic definitions like XML, this flexibility can significantly enhance interoperability without requiring perfect translations. Unfortunately, there is no single, simple solution to the problem of ensuring that two parties mean the same thing for a common term. This must be accomplished via human exchanges at design time. But EIDs can support this goal based upon the basic tenet that two items with the same EID must be semantically equal based upon the resolution of the definition.

[37] The terms field, column, attribute, and element are synonyms.

[38] For example, whether it is a relational DBMS (RDBMS), object DBMS (ODBMS), or flat file.

[39] Note that this is one of the weaknesses of typical data standardization efforts, such as the DoD 8320 process. A possible solution to the problem may be to adopt an approach where the '*data element*' is defined at a much higher level of abstraction, and its syntax can then be made explicitly part of it. The ISO 11179 draft recommendations present a possible solution to this issue.

This basic tenet can be exploited in (at least) three ways. First, a data source developer can provide several versions of the source. This could be based on technology (e.g., RDBMS, ODBMS, etc.), specialty (maneuver verses logistics), or language (English versus German). Items with the same EID are semantic equivalents, even if the format of the data maintained varies. Second, two independent source developers can collaborate and share EIDs for items that they agree are semantically equivalent. This approach is usually required for complex components and associations whose relationships are not obvious. Finally, and most commonly, a mapping can be built between EIDs for duplicative or overlapping entities that were independently defined.

3.7 Benefit To The Warfighter

One of the toughest challenges for warfighters is managing uncertainty. What normally comes to mind is the traditional "fog of war" associated with situational awareness. However, there are many other circumstances that require flexibility for which digital data systems do not respond well. Too often, digital systems are built with designs based upon strict process and data flow charts that describe each expected phase, input, output, source, and sink as a system (or system of systems) operates. Unfortunately, in the new environment of joint and coalition operations, one does not always know ahead of time with whom data will be shared and how it will be related. As the old adage goes: "a plan is only good until the first rounds are fired."

EIDs provide two important features to help the warfighter manage uncertainty. First, as previously described, they provide a uniform facility for relating together arbitrary pieces of data. During the design of a database system, only a very small percentage of the possible associations are identified. The vast majority of the set of all possible associations will never be used, so under the current database schemes it is unwarranted (and too costly in space, time, and money) to address all possible conditions. On the other hand, it is likely that many more associations will be required than were identified at system design time.

Since EIDs are uniform, unintelligent identifiers, a single table can be used to define any associate from the set of all possible associations. One does not need to identify the particular association ahead of time. This is because the format of the identifier is known, so an association can be established provided that an attribute the size of an EID is reserved. This is the ideal condition for a data mining system. Arbitrary relationships can be established without prior coordination. It is exactly this fundamental capability that is required to facilitate building data interoperability on-the-fly between joint, coalition, and non-military systems.

Figure 12 illustrates a simple example. A data structure named a "sensing" entity is established to associate three other entities: an owning organization, a sensor (a piece of materiel), and the object being sensed. In this example, this last association is maintained in a field called "Target." This field is the size (or format) of an EID, so it can be used to associate the sensing entity to *any* entity that is tagged with an EID. Suppose that a sensor makes an initial contact and determines that the item being sensed is a tracked vehicle. At state T_1, the sensing entity's target field contains the EID of the materiel-type[40] object for "tracked vehicle." As more information is gleaned, it is determined that the item being sensed is a particular type of tracked

[40] Also called a *materiel-item* by the logistics community.

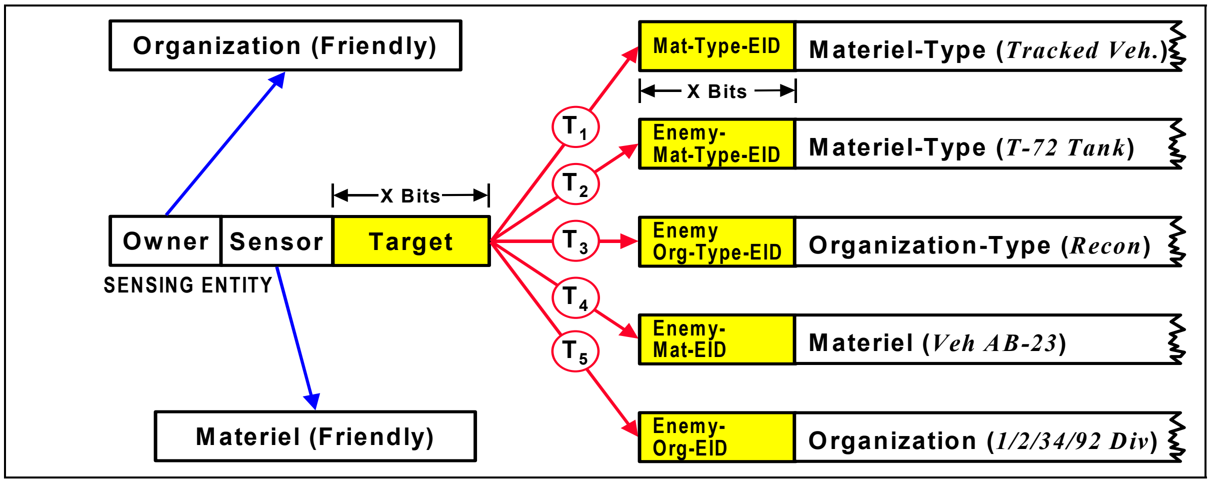


Figure 12: Arbitrary Plug and Play of References to Disparate Entities

vehicle, a T-72 tank. At state T_2, the Target field is updated with the EID for the enemy-materiel-type "T-72 Tank." A short time later at state T_3, it is determined that the vehicle is actually part of a cluster of vehicles that makes up a reconnaissance element, which is an organization-type. The EID in the Target field is updated to reference the organization-type entity of a recon element. Moving between entity types is not a problem because all EIDs have the same format allowing one to easily change the reference from a materiel-type to an organization-type. At state T_4, an actual vehicle identification number is determined, so the Target field is updated to hold the EID for a specific item of materiel (with a serial number). Finally, at state T_5, the identity of the cluster of vehicles is determined and the EID for a particular organization is entered in the Target field. Even though the entity being sensed progressed from a materiel-item, to an organization-type, to a piece of materiel, to an organization, only a single field (the Target field) was required to maintain all of these associations. One does not have to determine ahead of time all the possible cases. Because the Target attribute contains an EID, it can be used to denote an association with any entity in the enterprise: a person, waveform, bridge, location, network, plan, or anything else. This is the flexibility obtainable by using EIDs.

This is precisely the reason that the LC2IEDM[41] uses the *object* and *object–type* schema for the five basic battlefield objects. It allows any ORGANIZATION, PERSON, MATERIEL, FEATURE, or FACILITY (and their typed variants) to be associated. However, to accomplish this objective across arbitrary data systems, having a common format is not sufficient. One must also have unique enterprise identifiers (the 'E' in EID).

The second advantage to the warfighter is an "out-of-band" facility to provide unique identifiers across unanticipated connections. Anyone deemed potentially valuable to the enterprise can obtain an account on the ESS and independently implement EIDs. In other words, this is an

[41] See footnote 17, pg 6.

open standard—there is nothing proprietary or closed about it.[42] Neither the military commander or acquisition community has to enforce this action. Any country or organization that desires to exchange data with military systems can use EIDs based on EID seeds from the ESS. Because EIDs are unintelligent, they can be used by anyone. Because they do not have to be primary keys, they can be added at any time. Thus, the EID facility offers the warfighter a simple facility to substantially assist in the building of "on-the-fly" distributed systems.

3.8 Benefits To Database Evolution And Managing Change

A primary impetus for developing EIDs (and earlier surrogate keys) was system maintenance and evolution. Data interoperability is just one of many secondary effects realized by EIDs. EIDs provide a flexible approach to evolve the structure of a database and manage the inevitable changes that arise from new requirements. Cost saving benefits can be accrued when data is managed more like objects rather than tables in a traditional RDBMS. Instead of reviewing the extensive benefits of EIDs in this area, a series of excellent articles are recommended. From a relational perspective, Dr. Tom Johnston has written an excellent series of articles on enterprise keys (EK) for *DM Review* that are available online.[43] The first two articles describe the basic problem and the last four discuss implementation issues. Of particular interest is the *foreign key ripple effect* described in the articles. Other advantages to using unintelligent global keys are also presented that are of fundamental interest to DBMS designers and users.

In addition, because EIDs are technology independent, they offer a potential bridge between relational and object-oriented technologies. This chasm is very apparent in the Army between the command and control (C2) and simulation communities. A recent effort to accelerate interoperability between real systems and simulations has resulted in several programs to integrate these communities (see the Army's Simulation to C4I Systems Integration (SIMCI) project[44]).

[42] Current initiatives to adopt commercial systems with their proprietary ERP solutions may not be adequate to support the ad hoc end user of DoD information systems, namely the coalition warfighter.

[43] Tom Johnston has written an excellent set of articles on EKs for DM Review that is available online. The first two articles describe the basic problem and the last four discuss implementation issues:

Primary Key Reengineering Projects: The Problem; DM Review, February, 2000; http://www.dmreview.com/master.cfm?NavID=55&EdID=1866

Primary Key Reengineering Projects: The Solution; DM Review, March, 2000; http://www.dmreview.com/master.cfm?NavID=55&EdID=2004

De-Embedding Foreign Keys, Part 1: DM Direct, June 2, 2000. http://www.dmreview.com/editorial/dmreview/print_action.cfm?EdID=2308

De-Embedding Foreign Keys, Part 2; DM Direct, June 9, 2000. http://www.dmreview.com/editorial/dmreview/print_action.cfm?EdID=2322

De-Embedding Foreign Keys, Part 3: DM Direct, June 16, 2000. http://www.dmreview.com/editorial/dmreview/print_action.cfm?EdID=2331

De-Embedding Foreign Keys, Part 4: DM Direct, June 23, 2000. http://dmreview.com/editorial/dmreview/print_action.cfm?EdID=2341

[44] SIMCI: see https://simci.army.mil/

4. PERSONNEL ONTOLOGY AND MODELING ISSUES

4.1 Basic Entities in Battle Command System Models

To exchange information unambiguously, there must be agreement on the meaning of terms, concepts, and processes. For automation purposes, an agreement must be formal; that is, it must be methodically represented in a consistent form. However, there is no "correct" form for an agreement because one's perspective determines the details of the terms, concepts, and processes. This is certainly true between the administrative and battlefield systems that exchange personnel information.

The term "ontology" is used to describe how one represents the knowledge and information of a particular domain or subject area.[45] One way to specify an ontology is with a data model. A popular technique being used throughout the database community to specify relational database structures is IDEF1X.[46] Fortunately, both the administrative and battlefield system database designers use IDEF1X to document their models. Unfortunately, as expected, they have different perspectives of how to classify the information. Systematics is the science of classification and it includes the building of taxonomies of domains. This is one of the basic tasks of the modeling process and is accomplished via the generalization hierarchy in IDEF1X.

Recall from Section 3.1 that there are several battle command systems that utilize one of several related data models (all in IDEF1X notation) that incorporate five basic battlefield domains.[47] These domains have two basic classes of entities: instances and templates. Generally, instances are created from templates and the term "IS A" is often used for these associations. For example, "A Company is a Rifle Company" where A Company is the instance built from the rifle company template. Usually, there are many instances for a single template and (in these data models) the template name ends with the suffix "type." Figure 13 provides the names of the instances and the associated type for the five battlefield domains along with a simple example. The AOS will include four of these entities: organization, organization-type, personnel-type, and materiel-type and the default relationships between them.

It is important to understand the distinction between instances and types. Army force structure documents, like TOEs and MTOEs, refer to type information from which several real organizations (ORG) can be activated. In other words, organization-types (ORGT) provide a hierarchical structure of other ORGT that are associated with the types of people and materiel required to support a set of required missions.

[45] See http://www-ksl.stanford.edu/kst/what-is-an-ontology.html for a short description of ontologies.

[46] IDEF1X: **I**CAM (integrated computer aided manufacturing) **DEF**inition method number **1** – e**X**tended. For an excellent tutorial, see Thomas A. Bruce. *Designing Quality Databases with IDEF1X Information Models*. Dorset House Publishing, New York, NY, 1992.

[47] See footnotes 17 and 18, pp 10 and 11.

4.2 Personnel Data in the Context of Battle Command Systems Data Models

The personnel domain encompasses two of the ten entities described in Figure 13, persons and person-types. A person is "flesh and blood" while a person-type describes the qualifications and experiences of a person from a military perspective. It is important to understand the subtle differences between organization, organization-type, person (PERS), and person-type (PERST) entities used in the battle command system data models and how they relate to data entities in administrative data models. An understanding of their relationship with common terms (like billet, position, paragraph, line number, and SRC[48]) is also required. A description of these entities and terms is provided in the following sections.

4.2.1 Organization [ORG]

Organizations represent a real Department of Defense (DoD) unit with people assigned and an official unit name. In the DoD, official units are provided a Unit Identification Code (UIC). However, the granularity of UIC assignments is limited. Within in the Army, UICs are normally assigned down to the echelon of company (battery or troop). Units below that may be identified with derivative UICs (of which there are many forms), or other identifiers, such as the Unit Reference Number (URN) used by the VMF[49] messaging community to track units on the battlefield, often at the platform level. In a relational database, every data record is identified with a primary key. In battle command system databases, like the JCDB, the Org-ID serves this purpose for organization records. Soon, Org-IDs will be replaced with Org-EIDs.

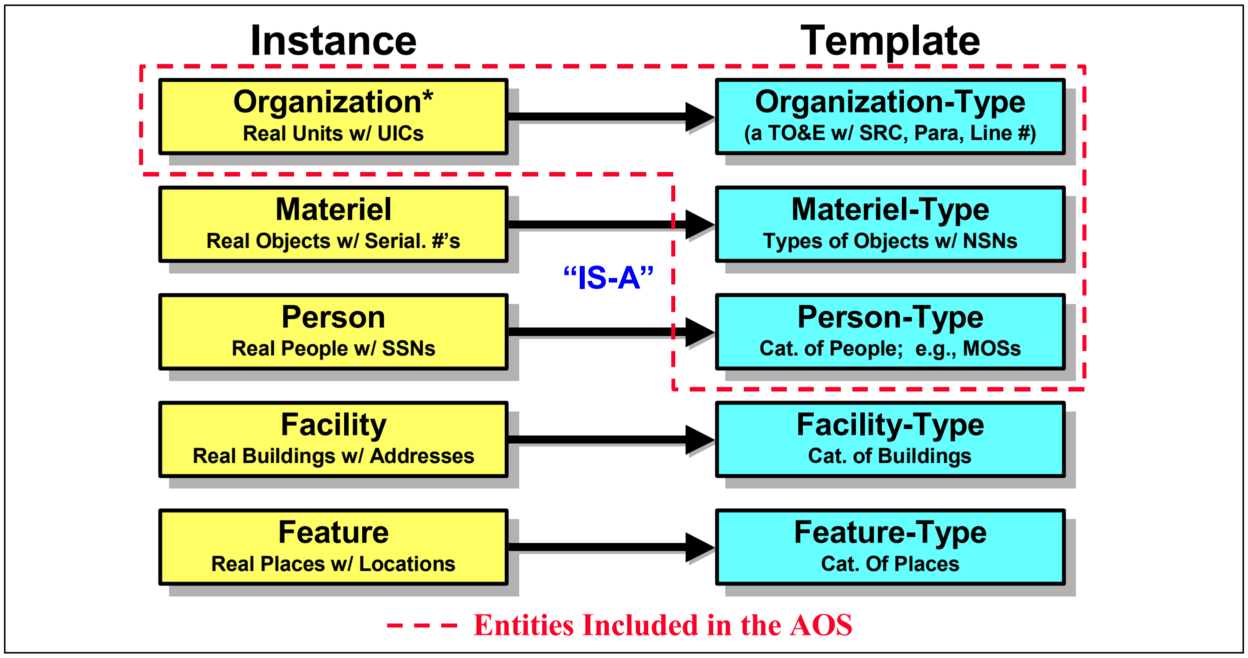


Figure 13: Examples of Instances and their Associated Templates

[48] SRC is a Standard Requirements Code used to identify TOEs. TOEs are updated annually (usually in April) in the Consolidated TOE Update (CTU). So there may be several versions of a TOE (with the same SRC) with different CTU effective dates. Therefore, an SRC and a CTU date are required to uniquely identify a TOE.

[49] VMF: Variable Message Format, a joint standard for passing formatted messages among tactical systems.

In the tactical environment, there is a requirement to identify organizations at a higher level of resolution than company (as the requirement for URNs has demonstrated). Consequently, a requirement for an AOS was generated and is included as part of the Force Management System (FMS) operational requirements document (ORD). FMS will migrate four existing mainframe systems to one modern system using a single integrated database to manage the active, reserve, and National Guard units that make up the U.S. Army Total Force.[50]

One of the objectives of the AOS is to consistently extend the current force structure hierarchy down to the billet level. A ***billet*** is "a personnel position or assignment that may be filled by one person."[51] In other words, a billet becomes just another organization, but with the additional constraint that it has one-to-one association with a person. Thus, the distinction between an SRC, paragraph, and line number (found in force structure documents) dissolves as all the members of the organization-type hierarchy are tagged with a consistent identifier, a primary key called an Org-ID. To further enhance maintainability and interoperability in the AOS, Org-IDs will use the EID structure. So technically, they are ORG-EIDs, but they are functionally identical to Org-IDs.

4.2.2 Organization-Type [ORGT]

Organizations [ORG] are instances of organization-types [ORGT], or conversely, organization-types represent the templates from which organizations are established. Consequently, the elements of Army TOEs and MTOEs are denoted by ORGTs, and not ORGs. Further, the nodes of a TOE or MTOE hierarchy do not get tagged with ORG-EIDs, but with ORGT-EIDs. This distinction between instances and their types may appear trivial, but it is often a source of confusion when discussing issues about this domain. It is easy to refer to ORGs when one means ORGT, and vice versa. This is especially true at the billet level.

If a billet is an ORG, then what is the corresponding entity in the ORGT domain? A convenient term to use would be a *billet-type*. However, in many personnel systems, the term ***position*** is used. Unfortunately, this term does not exist in the DoD dictionary, and the definition of billet is ambiguous: "a personnel position or assignment." In this definition, the words "personnel position" implies an ORGT, while the words "personnel assignment" and the phrase "filled by one person" seem to imply an ORG.

To further complicate matters, in the battle command system data models, the generalization hierarchies for ORG and ORGT use the terms POST and POST-TYPE (respectively) for the billet level organizational entities. These definitions are clearly defined; a "Squad Leader" is a POST-TYPE, while "Squad Leader/1st Squad" is a POST. Having multiple definitions, and closely related ones at that, infuses considerable confusion into this domain. To abide by the vernacular of the personnel community, it is recommended that the term **position** be used to represent templates (an ORGT) and the term "**billet**" be used for the actual instantiation of the template (an ORG). Under this paradigm, it is correct to state "a billet is authorized by a position."

[50] See: http://www.sra.com/services/fms.html

[51] See Joint Publication 1-02 (15 Oct 2001): http://www.dtic.mil/doctrine/jel/new_pubs/jp1_02.pdf.

A current example can be found in the SIDPERS-3 data model[52]. Positions are stored in a table named AUTH_POSN (for authorized position). Using the traditional relational approach (as opposed to EIDs), a record for an authorized position is identified with a primary key that includes three foreign keys: an authorization document key (AUTH_DOC_NBR) for an MTOE, a paragraph key (AUTH_DOC_PARA_NBR), and a line number key (AUTH_DOC_LINE_NBR). Together, these three keys uniquely identify an Army position (i.e., an ORGT) that is accompanied by a multiplier in an MTOE. The multiplier defines how many of the Army positions are authorized. For example, it may be stated that a rifle platoon is authorized three "Squad Leader" positions. When a real unit is instantiated via this authorization, the multiplier must be used to create three separate billets, each an ORG. In the SIDPERS-3 data model this is accomplished by using the UNIT-MANNING table that adds a fourth attribute called "MILITARY-ASSIGNMENT-POSITION-NUMBER" to differentiate each of the three squad leader billets. It is this organization, or billet, to which a soldier may be assigned.

Using the terms defined in Figure 1, this configuration results in two distinct categories of hierarchical tree structures, as is illustrated in Figure 14. The first category represents MTOE data, or the templates from which real organizations are created. The nodes of these trees are ORGTs and the links between the ORGTs include multipliers that define how many of each subordinate ORGT are authorized. An ORGT tree is shown on the left with nodes A, B, and C. The meaning of the links is aggregation and can be read as "Is-Composed-Of." Node B has a multiplier of 1 and node C has a multiplier of 3. So the chart is read: "node A is-composed-of one node B and three node C's.

The second category of tree represents real units that are established from the ORGT tree. The nodes of this tree are ORGs and a separate node must be instantiated for each multiple defined in the ORGT tree. An ORG tree is shown on the right with nodes 1 through 5. Like the ORGT tree, the links denote aggregation. So the ORG tree is interpreted as node 1 is-composed-of nodes 2 through 5. Because the ORG tree represents real units, one could also state "organization 1 has

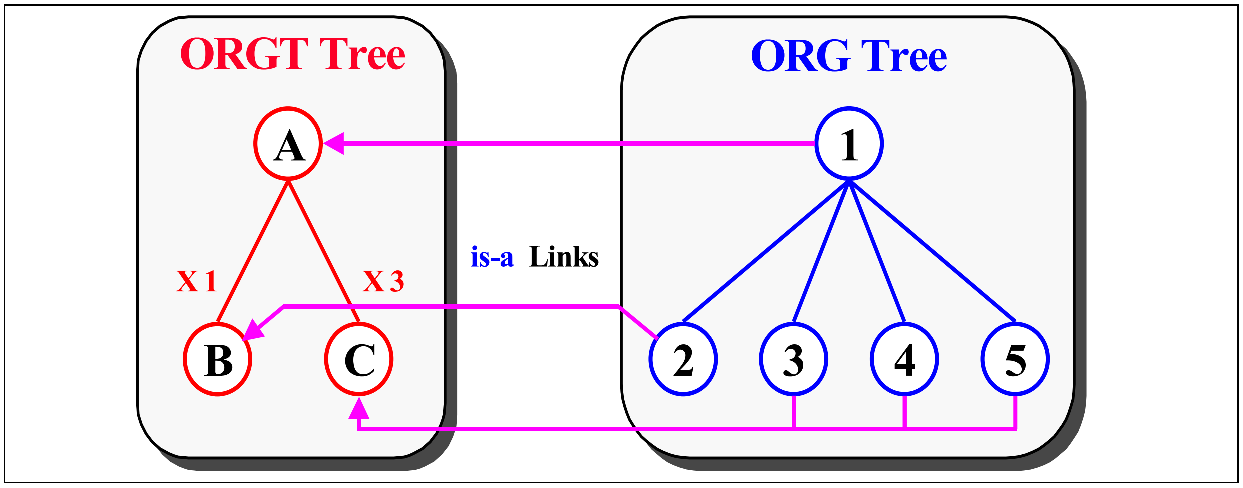


Figure 14: ORGT and ORG Trees

[52] SIDPERS-3: Standard Installation/Division Personnel System - Version 3, a prevalent Army personnel system that is now controlled by the Army Human Resource System Product Management Office; See: http://www.peostamis.belvoir.army.mil/index4.htm.

subordinate organizations 2 through 5."

Every ORG node must have a link to the corresponding ORGT node from which it was established. These are often named "is-a" links. It can be stated that "node 1 is–a node A," etc. In other words, many of the basic characteristics of a real organization are stored in its corresponding organization-type. Because many ORG trees may be established from a single ORGT tree (i.e., many units may be established from a single MTOE), these common characteristics need only be maintained in a single place – the ORGT tree. The five basic tables required to maintain this model are: the ORGT table (nodes) and the ORGT-ASSOCIATION table (links) for the ORGT tree; the ORG table (nodes) and the ORG-ASSOCIATION table (links) for the ORG tree; and the ORG-ORGT-ASSOCIATION table for the "is-a" links between the nodes of the ORG and ORGT trees.

Every row of these tables will have an EID that globally identifies that row. Consequently, there will be ORGT-EIDs, ORG-ASSOC-EIDs, ORG-EIDs, ORGT-ASSOC-EIDs, and ORG-ORGT-ASSOC-EIDs. It is the ORG-EIDs that are of particular interest to the tactical commander because these are the items that are used to track unit status.

4.2.3 Person [PERS].

A person represents a physical entity composed of flesh and blood and, within the ranks of the U.S. military, is assigned an SSN (Social Security Number). The primary key for PERS is a PERS-ID. One may ask why not use SSN? In battle command systems the answer is simple – a person can be friendly, enemy, neutral, or unknown, and in many cases may not have a (known) SSN. Therefore, SSN is included as an attribute, and in many cases is an alternate key, but is not the primary key.

4.2.4 Person-Type [PERST] and Skill-Type [SKILLT]

The person-type entity (PERST) already exists in several data models, whereas the skill-type entity (SKILLT) is limited to a few. The mapping of person-type information between battle command systems and administrative environments is more elusive than ORGT because (1) PERST is used for two different purposes, and (2) equivalent attributes are spread among different entities.

PERST has been used to describe both the *requirements* for a position and the *qualifications* of a person. In the first case, a PERST is associated with an ORGT (i.e., a position) and defines the skills and credentials required for a person to occupy the position. Any ORGT node may have links to define the types and number of people and materiel that are authorized for that ORGT. This is illustrated in Figure 15. In this case, an ORGT tree beginning with an ORGT named "M2A2 Vehicle Crew" is composed of three subordinate ORGTs that are positions. These positions are called "Section Leader," "Gunner," and "Driver." Each position has a corresponding PERST entity that describes the *requirements* to occupy that position. For example, to be a Section Leader, one must be of grade E-6, have MOS 11M30, and be qualified as a Master Gunner. Similarly, a materiel-type (MAT_TYPE) may be associated with any ORGT. In this example, each "M2A2 Vehicle Crew" is authorized a "M2A2 BFV" (Bradley Fighting Vehicle). Therefore, Figure 15 illustrates a subset of the information maintained in an MTOE that is used to establish real organizations.

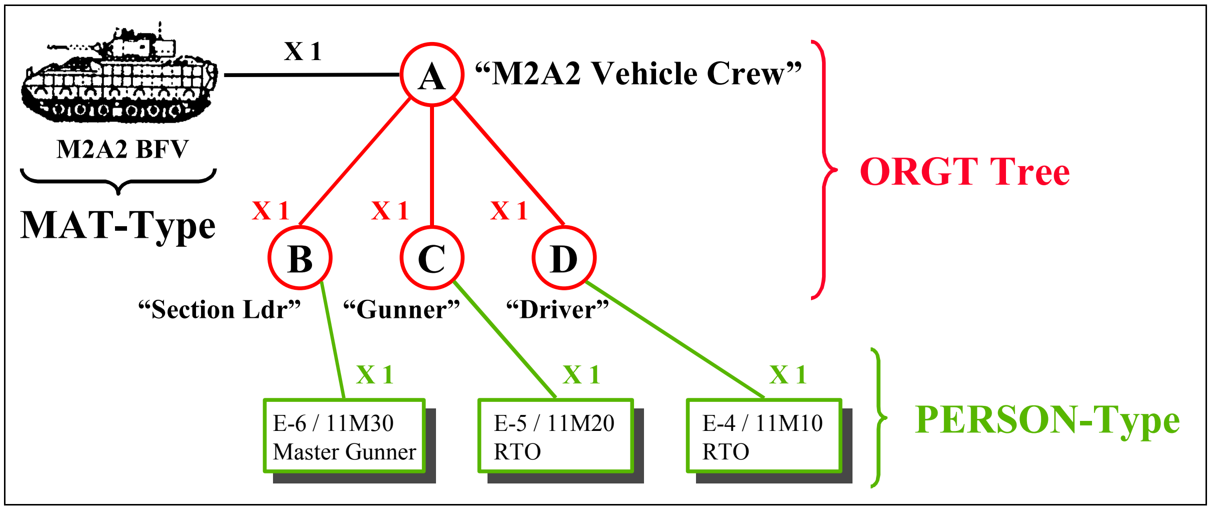


Figure 15: PERST Used to Define Requirements for an ORGT

In the second case, PERST modifies a PERSON and describes the skills and credentials earned by the person. This "overloading" of the use of PERST information is common across the services. Army, Navy, and Air Force personnel regulations describe the scope of their classification systems as applying to both the descriptions of the requirements for positions and to the qualifications of those who fill them. An example of these two uses is illustrated in Figure 16 for the LC2IEDM.

The path annotated by the number '1' illustrates the use per Figure 15. It denotes that an ORGANIZATION-TYPE (ORGT) is authorized one or more persons with the qualifications defined in the associated PERSON-TYPE. When the ORGT is a position (i.e., a POST-TYPE), the number of

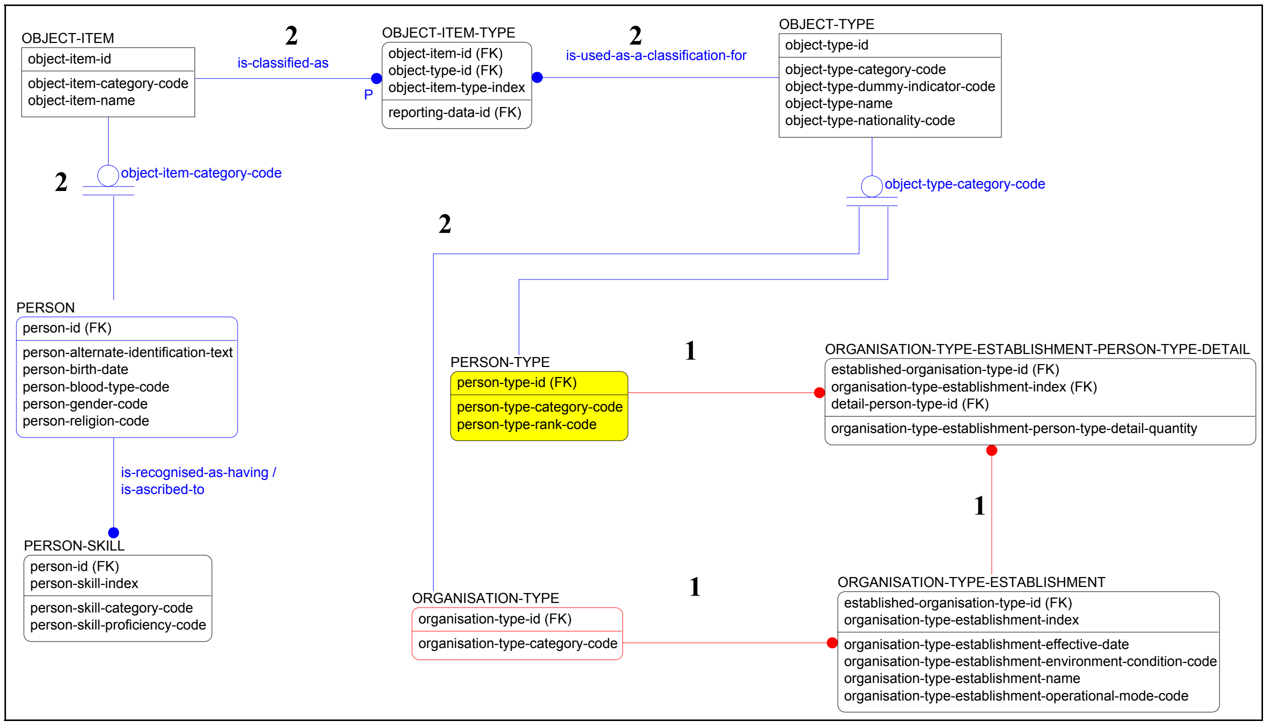

OBJECT-ITEM
object-item-id
object-item-category-code
object-item-name
2
is-classified-as
P
OBJECT-ITEM-TYPE
object-item-id (FK)
object-type-id (FK)
object-item-type-index
reporting-data-id (FK)
2
is-used-as-a-classification-for
OBJECT-TYPE
object-type-id
object-type-category-code
object-type-dummy-indicator-code
object-type-name
object-type-nationality-code
2
object-item-category-code
object-type-category-code
2
PERSON
person-id (FK)
person-alternate-identification-text
person-birth-date
person-blood-type-code
person-gender-code
person-religion-code
PERSON-TYPE
person-type-id (FK)
person-type-category-code
person-type-rank-code
1
ORGANISATION-TYPE-ESTABLISHMENT-PERSON-TYPE-DETAIL
established-organisation-type-id (FK)
organisation-type-establishment-index (FK)
detail-person-type-id (FK)
organisation-type-establishment-person-type-detail-quantity
is-recognised-as-having /
is-ascribed-to
1
PERSON-SKILL
person-id (FK)
person-skill-index
person-skill-category-code
person-skill-proficiency-code
ORGANISATION-TYPE
organisation-type-id (FK)
organisation-type-category-code
1
ORGANISATION-TYPE-ESTABLISHMENT
established-organisation-type-id (FK)
organisation-type-establishment-index
organisation-type-establishment-effective-date
organisation-type-establishment-environment-condition-code
organisation-type-establishment-name
organisation-type-establishment-operational-mode-code

Figure 16: LC2IEDM Associations with Person-Type

persons is one, so the PERST entity describes the qualifications required of that position. The reason the association is made with the more generic ORGT rather than a POST-TYPE is to allow aggregation at any echelon (e.g., a tank battalion is authorized ten officers of grade 0-3 and occupation code of 12A00).

The path annotated by the number '2' denotes that a PERSON has the qualifications defined in the associated PERSON-TYPE. In the ideal case, the qualifications of the person in the job match the qualifications required for the job. If this were the case, then the record of the person in the billet would refer to the same PERST as the record of the position associated with the billet; in other words, "the right person is in the right job."

Another common characteristic of personnel attributes that spans across the services is that, at a higher level, they can be divided into two categories: those that are mandatory and those that are optional. For example, every position (ORGT) and person entity must have a pay grade and occupation associated with it, even if the occupation is defined as immaterial.[53] These are mandatory attributes. But each service also has attributes that describe requirements of qualifications that span across occupations and pay grades that are not obligatory. These are optional attributes and have titles such as: "additional skill identifiers" (ASI) in the Army, "special experience identifiers" (SEI) in the Air Force, and "additional qualification designators" (AQD) in the Navy. Consequently, one strategy for categorizing attributes is to divide them into two categories, those that are mandatory and those that are optional, and create an entity to maintain each set. Should this approach be used, it is recommended that accepted names in the LC2IEDM, and other data models that have evolved from the ATCCIS Generic Hub, be exploited and that the PERSON-TYPE name be used for the entity containing the mandatory attributes and a new "skill-type" entity, SKILLT, be used for the optional attributes. Like PERST, SKILLT would be associated with both positions and persons as is illustrated in Figure 17. This allows the existing PERSON-SKILL entity of the LC2IEDM to be replaced with the more generic concept SKILL-TYPE.

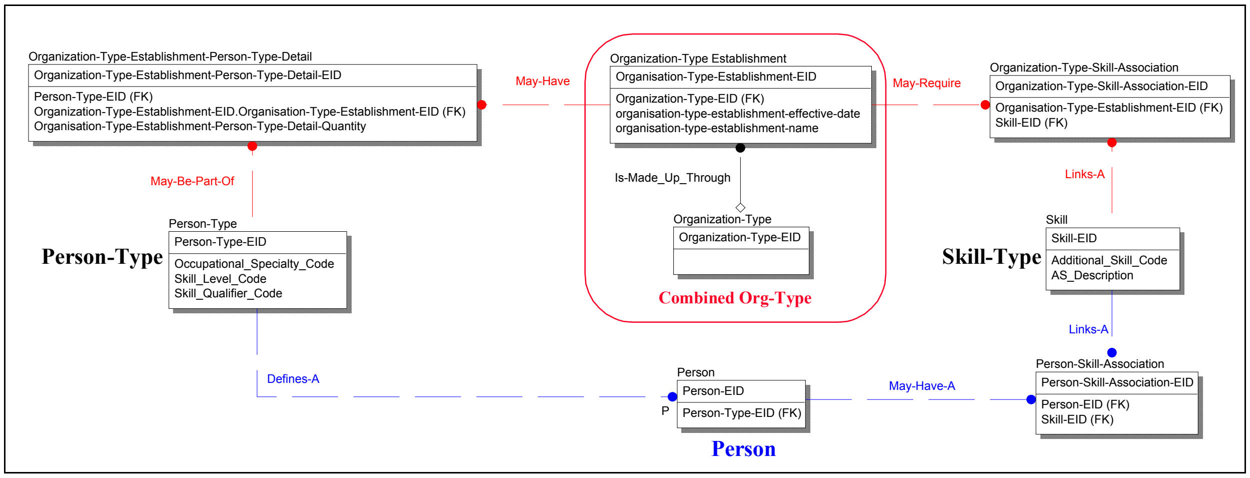

Organization-Type-Establishment-Person-Type-Detail
Organization-Type-Establishment-Person-Type-Detail-EID
Person-Type-EID (FK)
Organization-Type-Establishment-EID.Organisation-Type-Establishment-EID (FK)
Organisation-Type-Establishment-Person-Type-Detail-Quantity
May-Have
Organization-Type Establishment
Organisation-Type-Establishment-EID
Organization-Type-EID (FK)
organisation-type-establishment-effective-date
organisation-type-establishment-name
May-Require
Organization-Type-Skill-Association
Organization-Type-Skill-Association-EID
Organisation-Type-Establishment-EID (FK)
Skill-EID (FK)
May-Be-Part-Of
Is-Made_Up_Through
Links-A
Person-Type
Person-Type
Person-Type-EID
Occupational_Specialty_Code
Skill_Level_Code
Skill_Qualifier_Code
Organization-Type
Organization-Type-EID
Combined Org-Type
Skill-Type
Skill
Skill-EID
Additional_Skill_Code
AS_Description
Links-A
Defines-A
Person
Person-EID
P
Person-Type-EID (FK)
May-Have-A
Person-Skill-Association
Person-Skill-Association-EID
Person-EID (FK)
Skill-EID (FK)
Person

Figure 17: Person-Type, Skill-Type, Org-Type, and Person Associations

[53] Immaterial – meaning a position is open to a person of any occupation.

4.2.5 Attributes for PERST and SKILLT

There are numerous attributes associated with personnel entities, but for battle command systems the primary attributes of interest related to both requirement and qualification information are:

(1) rank and pay-grade,
(2) primary specialty or military occupation (e.g., MOS in the Army),
(3) special skill requirements (e.g., ASI in the Army).

The first two items, rank/pay-grade and military occupation, are *mandatory* in all the U.S. military personnel classification systems. This information is associated with every military position (ORGT) to define the requirements for the position, and with every person (PERSON) to define the person's basic qualifications. The third item, special skills, refers to *optional* information. A position or person may have significant additional skill information associated with it, or it may have none. For both position and personnel qualification data, it is recommended that the PERST entity be used to maintain mandatory information and the SKILLT entity be used to maintain optional information.

Using business rules, it can be stated that every ORGT entity that represents a position must have an associated PERST to provide the basic occupational requirement for the position. This is a general requirement for any force structure database. Further, for battle command systems, every military person entity must have an associated PERST entity to describe the basic qualifications obtained by the person. Both ORGT and PERSON entities may have SKILLT entities associated with them, but this is not required. A question that remains is how to divide up the attributes among the PERST and SKILLT entities.

Every service has a personnel classification system that rigorously defines many attributes. The concepts of *rank* (e.g., Master Sergeant or First Sergeant) and *grade* (e.g., E-8) are straightforward and grade is consistent across the services.[54] Unfortunately, this is not the case with military occupational data.[55] Each service uses different criteria to classify its position requirements and personnel qualifications. This is accomplished via a wide variety of *encoding schemes*, which makes decomposing the attributes into a common schema a difficult task. Even within a service there are significant variations in the way qualifications and requirements are encoded for officers versus enlisted personnel.

Consider the encoded position/qualification classification information for US military officers illustrated in Figure 18. One common characteristic among the services is that they all use ASCII characters to encode meaning into occupational codes. Further, the sequences of characters actually represent several different attributes. The values used for the codes yield little information; consequently, they are meaningful only to knowledgeable experts within the service. It is rare for anyone to recognize the occupational codes used by another service, let alone having a comprehensive knowledge of one's own service's codes. These examples illustrate just a few of the different personnel classification elements and encoding schemes used

[54] For a list see: http://www.defenselink.mil/pubs/almanac/almanac/people/insignias/
[55] For a list of occupational titles and associated coding see:
Army: https://www.odcsper.army.mil/pamxxi/secured/mosstructure/mos-charts.asp,
Navy: http://www.navmac.navy.mil/pubs.htm,
Air Force: http://www.afpc.randolph.af.mil/classification/

ARMY Officer: **Classification = "15C35/D7";**

Area of Concentration (AOC): 15C = Aviation All –Source Intelligence;
Functional Area (FA): 35 = Military Intelligence;
Additional Skill Identifier (ASI): D7 = AH-64D Pilot;

AIR FORCE Officer: **Classification = "K013B3B/OCE";**

Prefix: K = Instructor;
Air Force Specialty Code (AFSC): 013B3
013 = Space, Missile, and Command and Control;
B = Air Battle Manager;
3 = Aircraft commander qualified (specialty qualification level);
Specialty Shredout or *AF Specialty Title*: B = AWACS;
Special Experience Identifier (SEI):
Activity Code: O = Operations;
Experience Set: CE = Air Surveillance Officer;

NAVY Officer: **Classification = "1110/9364/0054T/LF7";**

Designator Code: 1110 = An Unrestricted Line Officer who is qualified in Surface Warfare
Navy Officer Billet Classification (NOBC): 9364 = Ship's Engineer Officer (Gas Turbine)
Subspecialty Code (SSP): 0054 = Naval/Mechanical Engineering;
T = Training billet which qualifies incumbent for an S-code (S = Significant experience).
Additional Qualification Designator (AQD) Code: LF7 = Officer designated as a Tactical Action Officer (with NTDS experience) IAW current instructions.

Figure 18: Examples of Officer Classification Schemes

by the different military services. Clearly, the techniques are diverse, but there are several ways to address this disparity.

One issue is the degree to which the character strings, or the fields they represent, should be maintained as individual components or simply left as aggregate strings. At one extreme, one can simply maintain a string of characters and let application programs parse the encoded strings into usable pieces. Table 1 illustrates how the three service examples of Figure 18 would appear using this option for occupation code. For each of the service examples, a PERST record has been created to hold mandatory information and a SKILLT record has been created for a single piece of optional information. The mandatory information includes rank, grade, and occupation data.

Table 1: The Occupation Attribute as a Single String

Example	PERST				SKILLT	
	EID	Rank	Grade	Occupation	EID	Skill
Army	0000000A	CPT	O-3	15C35	0000000D	D7
Air Force	0000000B	MAJ	O-4	K013B3B	0000000E	OCE
Navy	0000000C	LCDR	O-4	1110/9364	0000000F	0054T/LF7

Because rank and grade have a common meaning across the services, separate attributes for these characteristics are appropriate. However, occupation codes differ widely across the services, so one aggregated string is used for this attribute.

Recall that the information in Table 1 is used to describe both the requirements for a position (an ORGT) and the qualifications of a person. The record with EID 0000000A is a PERST entity that describes an Army officer of grade O-3 with the "15C35" occupational specialty. The SKILLT record with the EID 0000000D identifies the Army D7 skill. Because the occupation code is left as an aggregate set of symbols, the application program must contain the capability to extract the "15C" code and translate its meaning as an "Aviation All-Source Intelligence" qualified officer. Likewise, it must translate the "D7" skill code as meaning the ability to pilot an "AH-64D attack helicopter." Figure 19 illustrates occupation code aggregation using an IDEF1X representation.

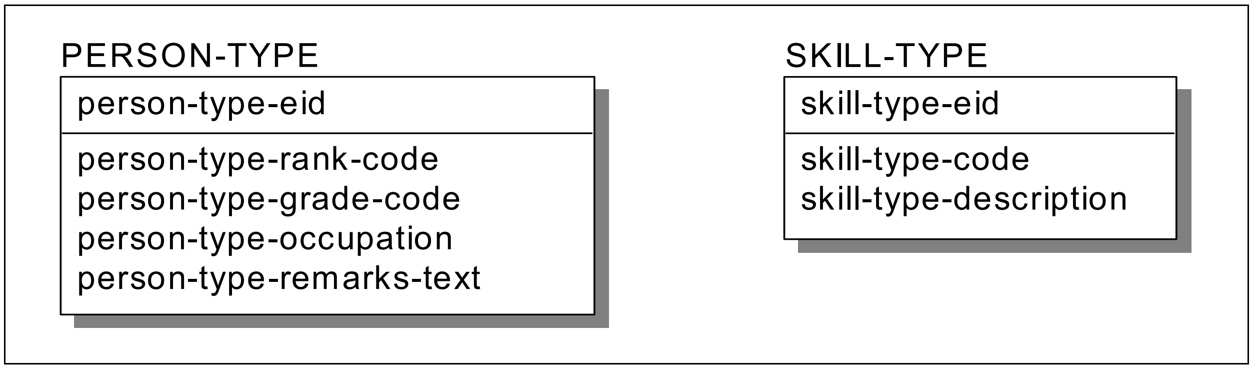


Figure 19: IDEF1X Representation of Aggregate Occupation Attribute

The primary advantage of this scheme is simplicity. Because the attributes are just character strings, data are easy exchanged between databases since each service (or country) hides its information within a collection of coded fields. But there are several disadvantages to this scheme. First, less usable information is maintained in the database. The applications are tasked with incorporating the knowledge of the encoding scheme to specify queries and filters. Rigor is not added to the database model through the enumeration of allowable database values. All these issues must be handled by the applications.

A second disadvantage to using a single, monolithic attribute is the combinatoric affect caused by the fact that every usable combination of occupation codes requires a different PERST. This is because nearly duplicate entries (e.g., two strings with all the characters the same except one) are different and require a different PERST. Third, because the codes are unfamiliar to the uninitiated, it may be desirable to include a descriptive title in the PERST entity. However, the aggregate character strings are composed of a set of concatenated codes, so one cannot automatically create an associated human readable title other than by concatenating the direct translations of the codes. For example, if the Army code "15C35/D7" is converted into a title using code translation and concatenation, one produces the title: "Aviation All–Source Intelligence, Military Intelligence, AH-64D Pilot." Developing a more natural description, such as "MI Qualified AH-64D Pilot," would require human intervention, which is an undesirably manpower intensive solution.

Clearly, there are other alternatives to code aggregation. At the other extreme, the strings can be decomposed into their individual fields and maintained as separate attributes. A more likely approach will exist somewhere between these extremes. The issues and ramifications of these decisions are addressed next.

4.2.6 Modeling Options for Personnel Attributes

If code aggregation is not used, the question remains: "In the process of data modeling, to what level and in what form should one decompose person-type information?" The extreme level of attribute decomposition is the full normalization of the data to the level of the atomic elements encoded in the schemes for every agency (e.g., military service). This allows each attribute to be enumerated so that a lookup table can be created to ensure only valid codes are entered. This minimizes combinatoric affects, but results in a data model that is more complex with little attribute reuse. Consider the following syntactic decomposition of the attributes used in the classification schemes of the Army, Navy, and Air Force (POSCO stands for the common term "position code"):[56]

```
POSCO [20] ::= Army_POSCO | Navy_POSCO | AF_POSCO;
   Army_POSCO [5] ::= A_Off_POSCO | A_WO_POSCO | A_Enl_POSCO;
      A_Off_POSCO [5] ::= AOC Alt_AOC;
         AOC [3] ::= AC Sub_AC | IM_AOC;
            Sub_AC [1] ::= alpha[1];
            AC [2] ::= BR | FA;
               BR [2] ::= num[2];
               FA [2] ::= num[2];
            IM_AOC [3] ::= ë0í num[1] alpha[1];
         Alt_AOC [2] := AC | ì00î;
      A_WO_POSCO [5] ::= WO_MOS WO_SQI;
         WO_MOS [4] ::= BR Sub_MOS;
            Sub_MOS [2] ::= num[1] alpha[1];
         WO_SQI [1] ::= alpha_num[1];
      A_Enl_POSCO [5] ::= A_ENL_MOS A_ENL_Skill_Lvl ENL_SQI;
         A_ENL_MOS [3] ::= CMF Sub_CMF;
            CMF [2] ::= num[2];
            Sub_CMF [1] ::= alpha[1];
         A_ENL_Skill_Lvl [1] ::= num[1];
         ENL_SQI [1] = alpha[1];
```

[56] This decomposition uses Backus-Naur Form, or BNF, a common, general format to describe computer languages. See http://cui.unige.ch/db-research/Enseignement/analyseinfo/AboutBNF.html. The symbol ::= means "is defined as" and the | symbol denotes "or". For this example, square brackets are introduced to indicate numbers of characters.

```
Navy_POSCO [12] ::= N_Off_POSCO | N_ENL_POSCO;
   N_OFF_POSCO [12] ::= N_Desig Pri_NOBC Opt_Sec_NOBC;
      N_Desig [4] ::= N_Desig_Cat_Code N_Desig_Suffix;
         N_Desig_Cat_Code ::= num[3];
         N_Desig_Suffix [1] ::= N_Billet_Suffix | N_Off_Suffix
            N_Billet_Suffix [1] ::= num[1]; /* 0,1,2 */
            N_Off_Suffix [1] ::= num[1];
      Pri_NOBC [4] ::= NOBC;
         NOBC [4] ::= num[4];
      Opt_Sec_NOBC ::= NOBC | <nil>;
   N_ENL_POSCO [11] ::= Rating Pri_NEC Sec_NEC;
      Rating [3] ::= alpha[3];
      Pri_NEC [4] ::= NEC | <nil>;
         NEC [4] ::= num[4];
      Sec_NEC [4] ::= NEC | <nil>;

AF_POSCO [7] ::= AF_Off_POSCO | AF_Enl_POSCO;
   AF_Off_POSCO [6] ::= AFSC | RI | SDI;
      AFSC [6] ::= Opt_Prefix AFSC Opt_Suffix;
         Opt_Prefix [1] ::= alpha[1] | <nil>;
         AFSC [4] ::= AFS Qual_Lvl;
            AFS [3] ::= Career_Grp Utilization_Fld Funct_Area;
               Career_Grp [1] ::= num[1];
               Utilization_Fld [1] ::= num[1];
               Funct_Area [1] ::= alpha[1];
            Qual_Lvl [1] ::= num[1];
         Opt_Suffix [1] ::= alpha[1] | <nil>;
      RI [4] ::= alphanum[4];
      SDI [4] ::= alphanum[4];
   AF_Enl_POSCO [7] ::= Opt_Prefix EAFSC Opt_Suffix
      EAFSC [5] ::= EAFS Skill_Lvl Specific_EAFS;
         EAFS [3] ::= Career_Grp Career_Fld CF_Sub;
            Career_Fld [1] ::= alpha[1];
            CF_Sub [1] ::= num[1];
         Skill_Lvl [1] ::= num[1];
         Specific_EAFS [1] ::= num[1] | <nil>;
```

This BNF example expands each encoded string down to its atomic elements. This activity is analogous to using the IDEF1X modeling process to build a generalization hierarchy of the PERST and SKILLT entities. This is illustrated in Figure 20 for a few levels of a hierarchy for an Army slice of the domain.

Because each service has different rules and vernacular regarding the personnel classification process, there are very few common attributes. In a generalization hierarchy, this is revealed by the absence of attributes in the intermediate hierarchies that are shared among the services. This exemplifies the challenge of obtaining agreements for a common set of attributes caused by the differences in the way the services describe personnel requirements and qualification.

As other organizations (or countries) add their definitions, the problem rapidly increases. This is because the table structure incorporates the attribute names. A change in a name requires a

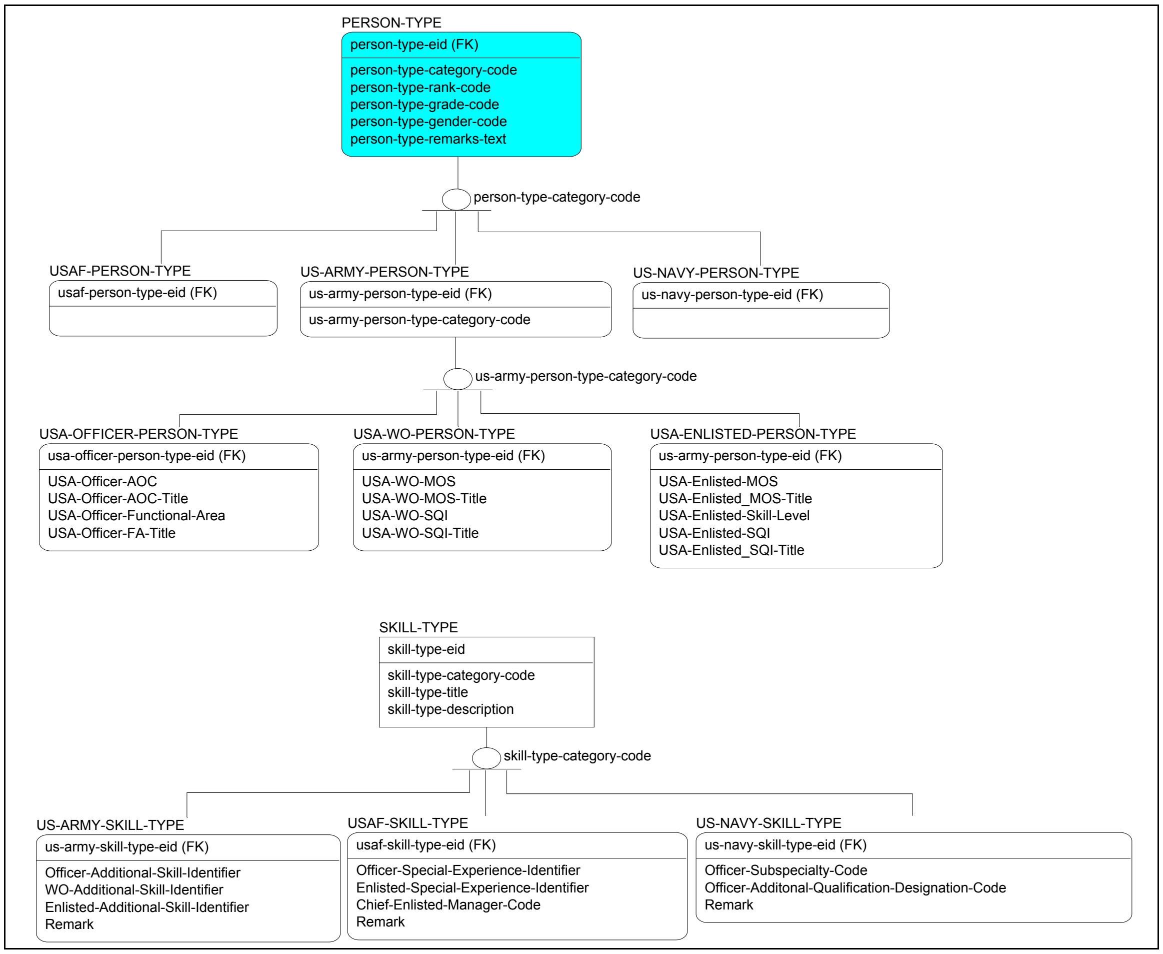


Figure 20: IDEF1X Representation of Decomposed Attributes

change to the database table structure. This is in contrast to the previous approach in which all the attribute information was maintained in the application programs. Clearly, the diversity of the person-type domain when using this approach (i.e., where every different personnel attribute has its own database table attribute) causes pragmatic issues in dealing with maintenance and change management.

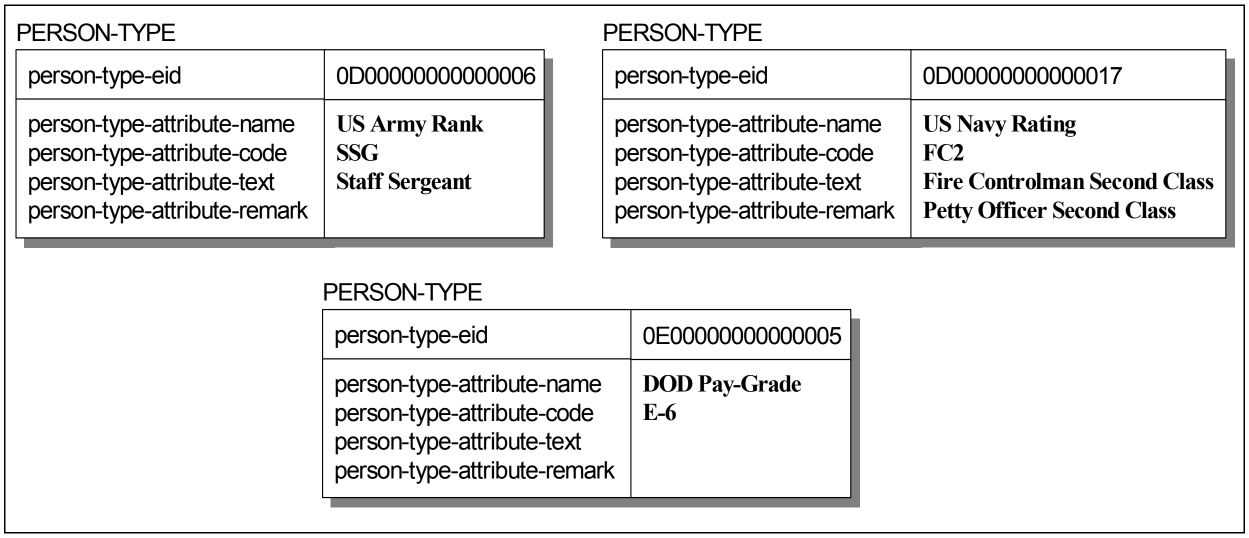
It is advantageous to find a compromise between the two extreme modeling methods presented thus far (character strings and generalization hierarchies). On one hand, it is advantageous to identify a general technique so that several parties can use the same attribute to store its encoding schemes. This facilitates the easy exchange of data between database systems and allows new participants to quickly join a consortium. When the codes are treated as simple ASCII text strings, new participants can enter their data about their person-types immediately. However, because a common semantics is not shared, the encoding schemes are only useful to those that understand them. Decoding must be accomplished either by the reader of the strings, or by application programs that have a knowledge of the encoding scheme incorporated within them. One could easily argue that this provides only marginal improvement in interoperability because the task was merely moved from the data model to the application.

It is desirable to make PERST information meaningful and useful to both experts and generalists alike. The distinction between expert and generalists not only refers to being inside and outside the personnel community, but also across personnel communities. For example, being a personnel expert in the U.S. Army (and thus knowing its encoding schemes) infers no familiarity with the U.S. Air Force, or any other, encoding scheme. A simple way to provide more useful information is to add descriptive text to the encoded text in the data model. In other words, to use two attributes — one to store encoded strings for those people and applications with the ability to interpret them, and another to provide generalists with descriptive information.

A challenge in incorporating these additional attributes is the automation of their insertion and maintenance. One does not want to create yet another set of attributes for humans to maintain. Therefore, it is advantageous to use existing data to create the set of entities to be transferred to battle command systems. The problem encountered is that the descriptive text desired is associated one-to-one with the atomic elements of the encoded strings, not with the string as a whole. There is no text associated with an aggregate code. For example, as illustrated previously, the Army code "15D31/D7" is actually a set of three codes, 15D, 31, and D7, for which each has an associated text description. There is no description for the aggregated set of all three. Therefore, one is restricted to developing a scheme that uses the atomic attributes in some manner. The challenge is to create a simple approach that is easy to use and maintain.

One technique to consider is using generic attributes to store descriptive names of specific personnel attributes. This is contrary to the technique illustrated in Figure 20 where a separate entity is used to contain the attribute names corresponding to each encoding scheme. Figure 21 illustrates this generic case. Instead of having attributes named "rank" and "grade," these attribute names are entered as data in a generic attribute called "attribute name." For the expert, an attribute called "attribute code" is provided, and for the uninitiated, two attributes called "attribute text" and "attribute-remark" are provided. For example, to find all the rank codes for the U.S. Army, one would execute a query on the attribute "person-type-attribute-name" with a value equal to the character string "US Army Rank."

This technique, called *generic attributes* for this discussion, converts each attribute from a column of a table into rows of a table. The advantage of this approach is its simplicity and flexibility. It allows as many or few attributes as required to be associated with an ORGT or PERS. It also allows easy insertion or deletion of new attributes without modifying the database



PERSON-TYPE

person-type-eid	0D00000000000006
person-type-attribute-name	**US Army Rank**
person-type-attribute-code	**SSG**
person-type-attribute-text	**Staff Sergeant**
person-type-attribute-remark	

PERSON-TYPE

person-type-eid	0D00000000000017
person-type-attribute-name	**US Navy Rating**
person-type-attribute-code	**FC2**
person-type-attribute-text	**Fire Controlman Second Class**
person-type-attribute-remark	**Petty Officer Second Class**

PERSON-TYPE

person-type-eid	0E00000000000005
person-type-attribute-name	**DOD Pay-Grade**
person-type-attribute-code	**E-6**
person-type-attribute-text	
person-type-attribute-remark	

Figure 21: Generic Attributes in Entity

structure. Further, it allows one to remove the SKILLT entity since the PERST entity can easily perform the role of both. If the mandatory versus optional characteristic of PERST versus SKILLT is required, then this distinction can be accomplished via another generic attribute, say “person-type-attribute-status.” The disadvantage of this approach is that the attribute name provides no additional information to the user about the data. For example, the structure provides no hint that the term “rating” in the Navy has a corresponding meaning to “rank” in the Army. Only by recognizing that all Navy ratings and Army ranks also have a common, sibling attribute named “grade,” would one be able to infer this useful fact.

An alternative technique is to incorporate limited bundling of attributes into entities. In this case, the distinction between mandatory PERST and optional SKILLT entities can be maintained. The size of the bundle must be decided and it can easily be increased over time. A cursory analysis of Army, Navy, and Air Force attributes resulted in the initial set of attributes illustrated in Figure 22 (with examples from Army and Navy enlisted manpower documents).

PERSON-TYPE

person-type-eid	0D00000000000001	0D00000000000010
person-type-rank-code	**SSG (Staff Sergeant)**	**FC2 (Fire Controlman Second Class)**
person-type-pay-grade-code	**E-6**	**E-6**
person-type-primary occupation-code-name	**USA Enlisted MOS Prefix**	**US NEC**
person-type-primary-occupation-code	**11M**	**1169**
person-type-primary-occupation-text	**Fighting Vehicle Infantryman**	**HARPOON (AN/SWG-1A) Maintenance Technician**
person-type-secondary occupation-code-name	**USA Enlisted SQI**	**US NEC**
person-type-secondary-occupation-code	**G**	**9595**
person-type-secondary-occupation-text	**Ranger**	**Hazardous Material Control Management Technician**
person-type-skill-level	**3**	-
person-type-remarks-text	-	-

Figure 22: Bundling Attributes into Entities

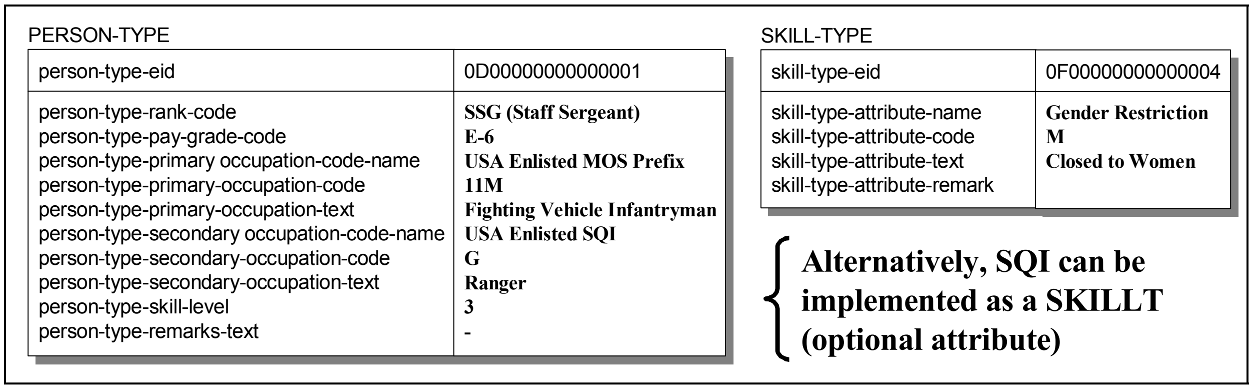
In this example, three principle categories of attributes are used: rank and grade, primary and secondary occupational information, and skill-level. These were chosen based upon a quick study of Army, Navy, and Air Force manpower documents.[57] All three services had these categories in their manpower documents. Foreign military services were not studied.

This technique, called *bundled attributes*, has two desirable features over unbundled, generic attributes. First, several, common, personnel attributes are conveniently bundled into a single entity. Second, the attributes are provided names with some additional meaning. They are not specific like those in the generalization hierarchy, but they are not void of information like generic attributes. In other words, they are a compromise. This allows a user to compare values across domains based upon the meaning inferred from a common attribute name (e.g., “primary occupation”). For example, it may be inferred that the enlisted military occupational specialty, or MOS, in the Army serves roughly the same purpose as the Navy Enlisted Classifications, or NEC, in the Navy. Further, both code and text fields are available for use by experts and generalists. A person knowledgeable in Army personnel techniques can easily compose an Army position code (POSCO) by combining three of the attributes – for example, the POSCO 11M3G.

[57] Army MTOE, Navy SMD (Ship Manpower Document), and Air Force UMD (Unit Manpower Document).

The corresponding disadvantage is the reduction in flexibility and increase in redundancy. For example, the word "rank" in the attribute "person-type-rank-code" is used generically. The only way one would know that the term is actually "rating" in the Navy (not rank) is by adding a comment in the remarks field. Further, if a third occupational specialty were required, another set of attributes would have to be added to the entity. Finally, some denormalization is introduced. For example, the value for pay-grade (e.g., E-6) is included in every entity instead of having a single entity for pay-grade that everyone uses.

Notice that the earlier distinction between PERST (for mandatory attributes) and SKILLT (for optional attributes) can be retained using this approach. The SKILLT entity provides the flexibility and simplicity offered using generic attributes while the PERST offers the advantages of using bundled attributes. Also recall that there is a one-to-one mapping to a PERST while any number of SKILLT entities can be associated with an ORGT or PERS. This allows both general and specific requirements (or qualifications) to be defined and used across entity boundaries. This combination of PERST and SKILLT is illustrated in Figure 23. This SKILLT entity is used to identify positions that are restricted to the male gender and can be used for any position (ORGT) in the database, regardless of service or country.



PERSON-TYPE

person-type-eid	0D00000000000001
person-type-rank-code	**SSG (Staff Sergeant)**
person-type-pay-grade-code	**E-6**
person-type-primary occupation-code-name	**USA Enlisted MOS Prefix**
person-type-primary-occupation-code	**11M**
person-type-primary-occupation-text	**Fighting Vehicle Infantryman**
person-type-secondary occupation-code-name	**USA Enlisted SQI**
person-type-secondary-occupation-code	**G**
person-type-secondary-occupation-text	**Ranger**
person-type-skill-level	**3**
person-type-remarks-text	-

SKILL-TYPE

skill-type-eid	0F00000000000004
skill-type-attribute-name	**Gender Restriction**
skill-type-attribute-code	**M**
skill-type-attribute-text	**Closed to Women**
skill-type-attribute-remark	

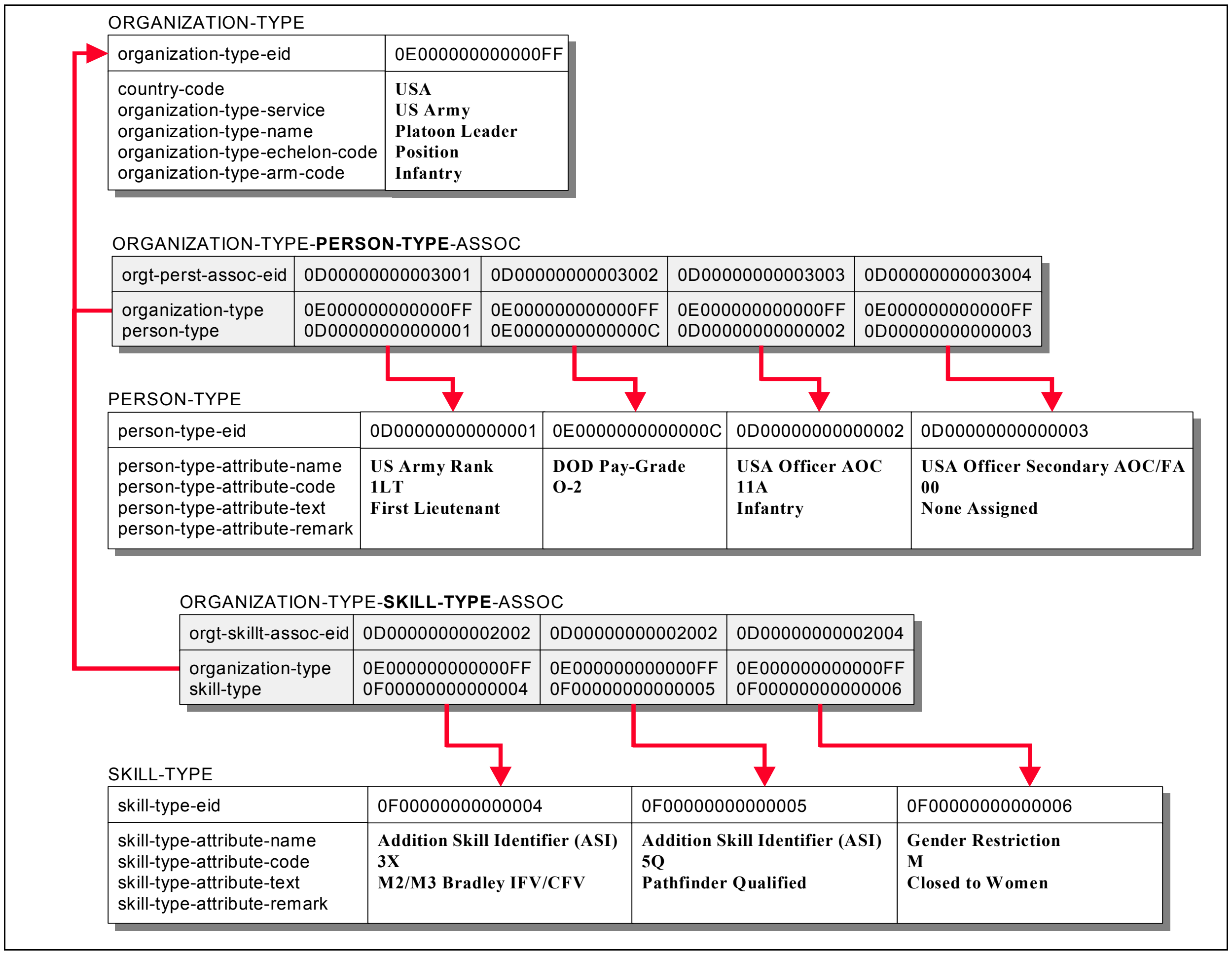
Alternatively, SQI can be implemented as a SKILLT (optional attribute)

Figure 23: Generic Skill-Type With Bundled Person-Type

4.2.7 Using EIDs to Add Flexibility to Associations

As with any relational database systems, relationships between entities are established using association tables. A primary advantage of EIDs is the simplicity and flexibility they provide when implementing physical database tables. Because all EIDs are the same size and format, a single format can be used to maintain many different types of associations. Figure 24 demonstrates the use of association tables with entities that use generic attributes. In typical relational style, there is a separate association table for each distinct pair of entities. In this example, there is one association table to relate ORGT with PERST and another to relate ORGT with SKILLT. Note that the association table may be composed of only EIDs.

By using generic attributes, each row of the PERST and SKILLT table contains a single characteristic. There is an entry in the association table for each appropriate characteristic. The fact that PERST attributes are mandatory and SKILLT attributes are optional is buried in the business rules that describe these entities. To obtain the mandatory attributes for the ORGT Platoon Leader, tagged with EID `0E000000000000FF`, an application program would execute a query on the ORGANIZATION-TYPE-PERSON-TYPE-ASSCOCIATION table for all rows that contain


ORGANIZATION-TYPE

organization-type-eid	0E000000000000FF
country-code organization-type-service organization-type-name organization-type-echelon-code organization-type-arm-code	**USA** **US Army** **Platoon Leader** **Position** **Infantry**

ORGANIZATION-TYPE-**PERSON-TYPE**-ASSOC

orgt-perst-assoc-eid	0D00000000003001	0D00000000003002	0D00000000003003	0D00000000003004
organization-type person-type	0E000000000000FF 0D00000000000001	0E000000000000FF 0E0000000000000C	0E000000000000FF 0D00000000000002	0E000000000000FF 0D00000000000003

PERSON-TYPE

person-type-eid	0D00000000000001	0E0000000000000C	0D00000000000002	0D00000000000003
person-type-attribute-name person-type-attribute-code person-type-attribute-text person-type-attribute-remark	**US Army Rank** **1LT** **First Lieutenant**	**DOD Pay-Grade** **O-2**	**USA Officer AOC** **11A** **Infantry**	**USA Officer Secondary AOC/FA** **00** **None Assigned**

ORGANIZATION-TYPE-**SKILL-TYPE**-ASSOC

orgt-skillt-assoc-eid	0D00000000002002	0D00000000002002	0D00000000002004
organization-type skill-type	0E000000000000FF 0F00000000000004	0E000000000000FF 0F00000000000005	0E000000000000FF 0F00000000000006

SKILL-TYPE

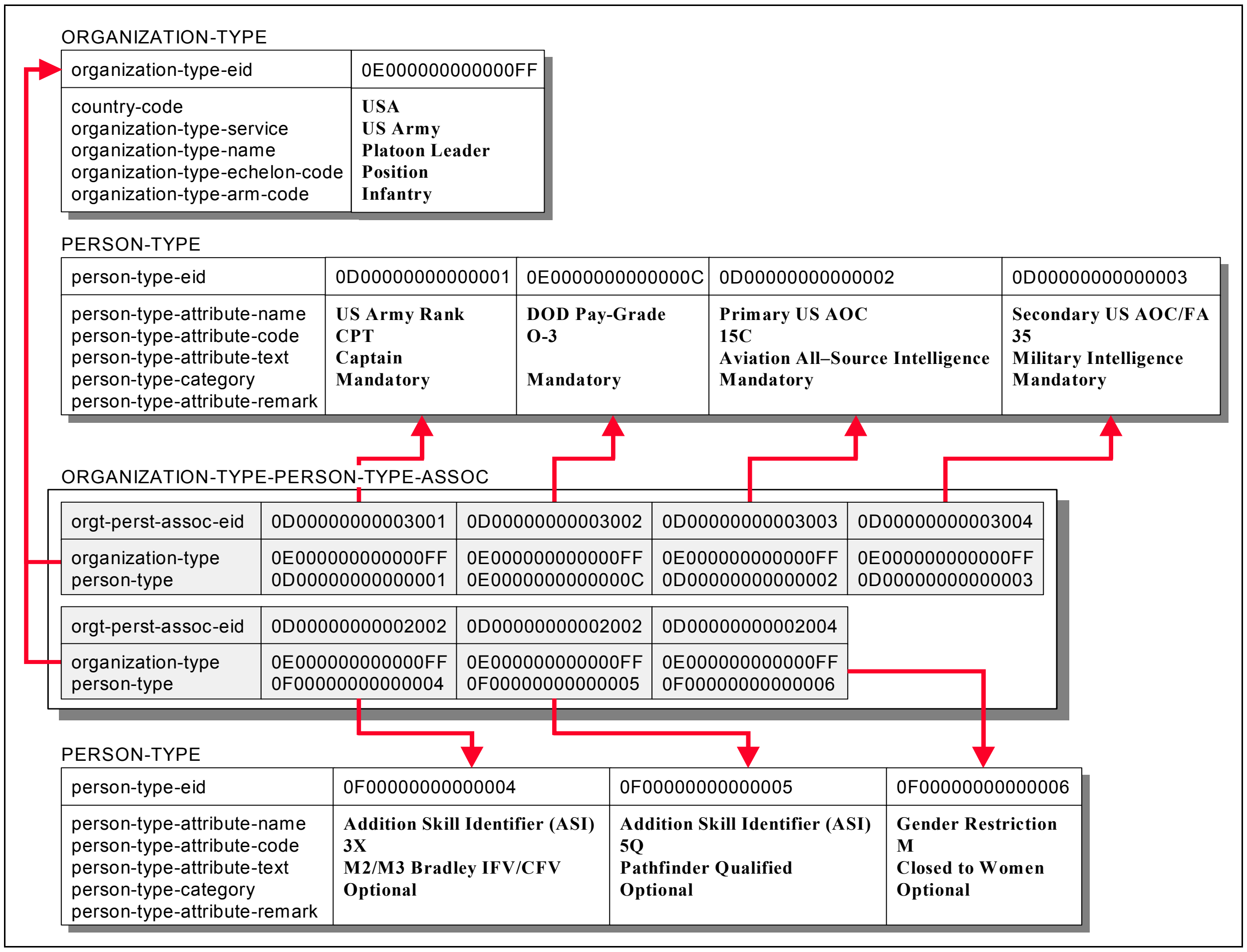
skill-type-eid	0F00000000000004	0F00000000000005	0F00000000000006
skill-type-attribute-name skill-type-attribute-code skill-type-attribute-text skill-type-attribute-remark	**Addition Skill Identifier (ASI)** **3X** **M2/M3 Bradley IFV/CFV**	**Addition Skill Identifier (ASI)** **5Q** **Pathfinder Qualified**	**Gender Restriction** **M** **Closed to Women**

Figure 24: Association Tables with Generic Attributes

that EID in the organization-type field. To obtain the optional attributes, a similar query is executed on the ORGANIZATION-TYPE-PERSON-TYPE-ASSCOCIATION table.

This illustrates the unwarranted use of both a PERST and SKILLT table when using generic attributes because they are identical in structure. The only distinction is the definition that PERST attributes are mandatory and SKILLT attributes are optional. By adding a new generic attribute to the PERST entity called "person-type-category" to contain the mandatory/optional distinction, the SKILLT entity can be eliminated. As illustrated in Figure 25, this action simplifies the data model by reducing the attribute and association tables to one each.

Bundled attributes represent a compromise between the extremes of generic attributes and full generalization hierarchies. Figure 26 illustrates the same example as Figure 24, but using a PERST entities composed of bundled attributes. For many people, this portrays a more familiar approach for the PERST entity. As in Figure 24, two attribute and association tables are required. The PERST entity contains mandatory attributes and the SKILLT entity continues to utilize generic attributes to define the optional attributes. However, the names used for the bundled attributes in the PERST entity now include a limited description of the type of information stored in the attribute. In Figure 26, four classes of information that are common and often mandatory across the U.S. military services are included: rank and grade, primary and secondary occupational



ORGANIZATION-TYPE

organization-type-eid	0E000000000000FF
country-code organization-type-service organization-type-name organization-type-echelon-code organization-type-arm-code	**USA** **US Army** **Platoon Leader** **Position** **Infantry**

PERSON-TYPE

person-type-eid	0D00000000000001	0E0000000000000C	0D00000000000002	0D00000000000003
person-type-attribute-name person-type-attribute-code person-type-attribute-text person-type-category person-type-attribute-remark	**US Army Rank** **CPT** **Captain** **Mandatory**	**DOD Pay-Grade** **O-3** **Mandatory**	**Primary US AOC** **15C** **Aviation All–Source Intelligence** **Mandatory**	**Secondary US AOC/FA** **35** **Military Intelligence** **Mandatory**

ORGANIZATION-TYPE-PERSON-TYPE-ASSOC

orgt-perst-assoc-eid	0D00000000003001	0D00000000003002	0D00000000003003	0D00000000003004
organization-type person-type	0E000000000000FF 0D00000000000001	0E000000000000FF 0E0000000000000C	0E000000000000FF 0D00000000000002	0E000000000000FF 0D00000000000003

orgt-perst-assoc-eid	0D00000000002002	0D00000000002002	0D00000000002004
organization-type person-type	0E000000000000FF 0F00000000000004	0E000000000000FF 0F00000000000005	0E000000000000FF 0F00000000000006

PERSON-TYPE

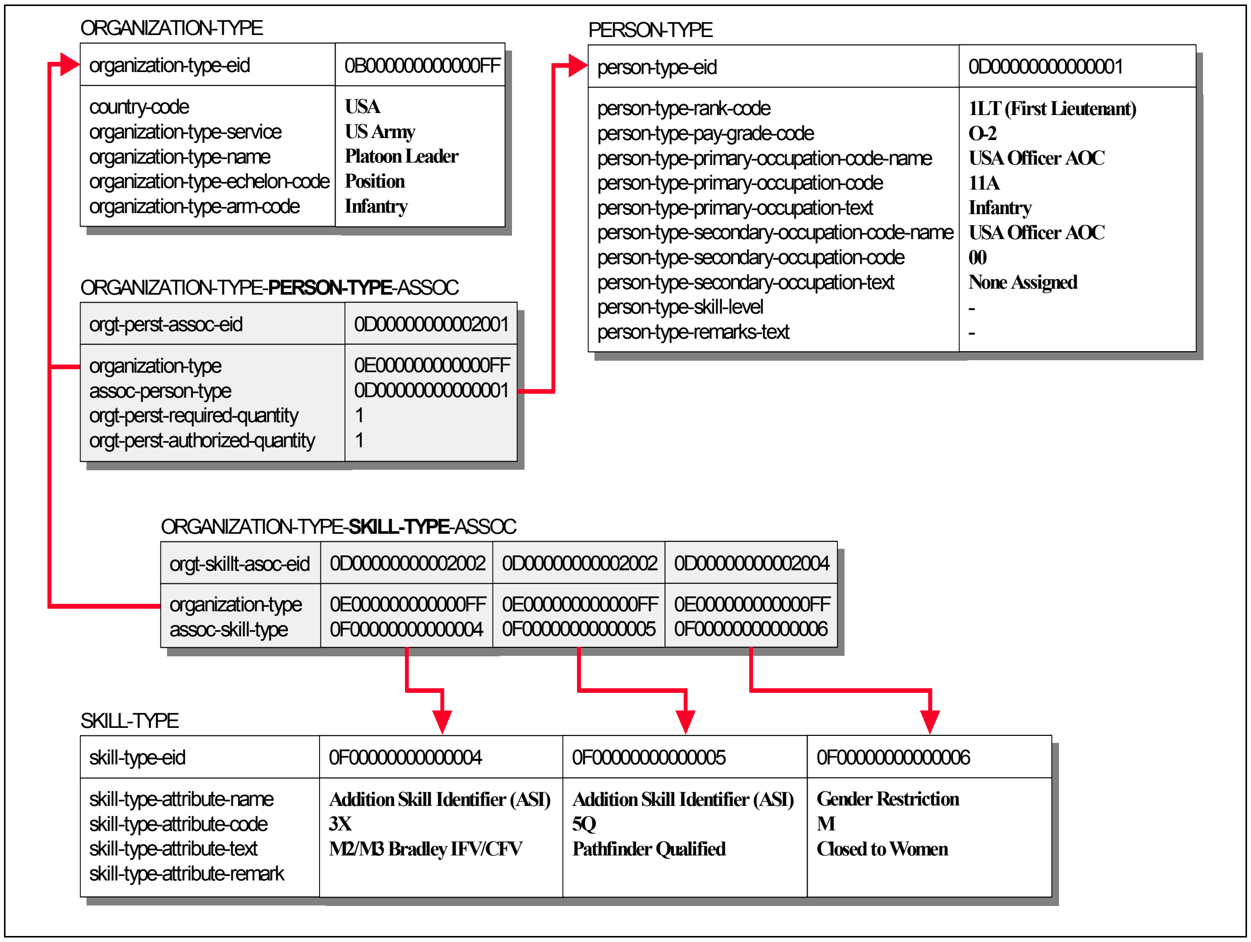
person-type-eid	0F00000000000004	0F00000000000005	0F00000000000006
person-type-attribute-name person-type-attribute-code person-type-attribute-text person-type-category person-type-attribute-remark	**Addition Skill Identifier (ASI)** **3X** **M2/M3 Bradley IFV/CFV** **Optional**	**Addition Skill Identifier (ASI)** **5Q** **Pathfinder Qualified** **Optional**	**Gender Restriction** **M** **Closed to Women** **Optional**

Figure 25: Using a Single Entity for Generic Attributes

specialty, skill-level, and a remarks field. Although some fields may be null, this approach provides the most used information in a single entity.

A drawback with bundling attributes is a reduction in normalization. When attributes are bundled, they may appear in many entities. In this example, all of the values in the fields of the PERST entity will appear in different combination in many other entities. If any of these values change, a global query will have to be executed to find all the instances that use the value so that it can be consistently and uniformly modified. This, of course, is a primary reason behind normalization; to ease maintenance, a specific piece of data exists in only one place in the database.

A simple way to alleviate this problem is to place all personnel attributes in tables and refer to them via their EIDs. An ideal choice is the SKILLT table, which allows all the personnel related attributes to reside in a single table to facilitate maintenance. This approach, called *bundled skill-types*, is illustrated in Figure 27, which uses the same example provided in Figure 26. The two SKILL-TYPE tables in the diagram represent different portions of the same table. Bundling SKILLTs simplifies the data schema is several ways. First, all the personnel descriptions can be found in a single table with common, generic attributes to maximize reuse. Second, the number of attributes of the PERST entity is reduced to the fundamental categories of rank and grade, primary and secondary occupation, and skill-level. Finally, there are no mandatory or optional



ORGANIZATION-TYPE

organization-type-eid	0B000000000000FF
country-code	**USA**
organization-type-service	**US Army**
organization-type-name	**Platoon Leader**
organization-type-echelon-code	**Position**
organization-type-arm-code	**Infantry**

PERSON-TYPE

person-type-eid	0D00000000000001
person-type-rank-code	**1LT (First Lieutenant)**
person-type-pay-grade-code	**O-2**
person-type-primary-occupation-code-name	**USA Officer AOC**
person-type-primary-occupation-code	**11A**
person-type-primary-occupation-text	**Infantry**
person-type-secondary-occupation-code-name	**USA Officer AOC**
person-type-secondary-occupation-code	**00**
person-type-secondary-occupation-text	**None Assigned**
person-type-skill-level	-
person-type-remarks-text	-

ORGANIZATION-TYPE-**PERSON-TYPE**-ASSOC

orgt-perst-assoc-eid	0D00000000002001
organization-type	0E000000000000FF
assoc-person-type	0D00000000000001
orgt-perst-required-quantity	1
orgt-perst-authorized-quantity	1

ORGANIZATION-TYPE-**SKILL-TYPE**-ASSOC

orgt-skillt-asoc-eid	0D00000000002002	0D00000000002002	0D00000000002004
organization-type	0E000000000000FF	0E000000000000FF	0E000000000000FF
assoc-skill-type	0F00000000000004	0F00000000000005	0F00000000000006

SKILL-TYPE

skill-type-eid	0F00000000000004	0F00000000000005	0F00000000000006
skill-type-attribute-name	**Addition Skill Identifier (ASI)**	**Addition Skill Identifier (ASI)**	**Gender Restriction**
skill-type-attribute-code	**3X**	**5Q**	**M**
skill-type-attribute-text	**M2/M3 Bradley IFV/CFV**	**Pathfinder Qualified**	**Closed to Women**
skill-type-attribute-remark			

Figure 26: Bundled Attributes and Association Table

personnel attributes. In essence, mandatory personnel attributes are those SKILLTs that are referred to within the PERST entity.

Two association tables still exist to relate SKILLTs with ORGTs: one for direct associations (the ORGANIZATION-TYPE-SKILL-TYPE-ASSOC table), and one to link the bundled SKILLTs via the PERST entity (ORGANIZATION-TYPE-PERSON-TYPE-ASSOC table). This means that, technically, one does not need to use the PERST entity under this approach. However, the purpose of the PERST entity is to add information about the personnel attributes commonly used across service boundaries so that comparisons can be made between them (e.g., service rank codes). Therefore, it is likely that entries will exist in both association tables to define the required qualifications for a position. The personnel attributed called mandatory are maintained via the PERST entity while those that were called optional are maintained via a direct relationship.

Thus far, the discussion of personnel data has assumed the special case of an association with a single person or position. Every PERSON entity that represents a military service member and every ORGT entity that represents a position must have a single PERST entity associated with it. This is how descriptions of personnel qualifications are maintained. However, this is not the general case for ORGT entities. Positions are just one of the three categories of ORGTs, the other two being crew/platform and doctrinal ORGTs. In these cases, it may be required to maintain

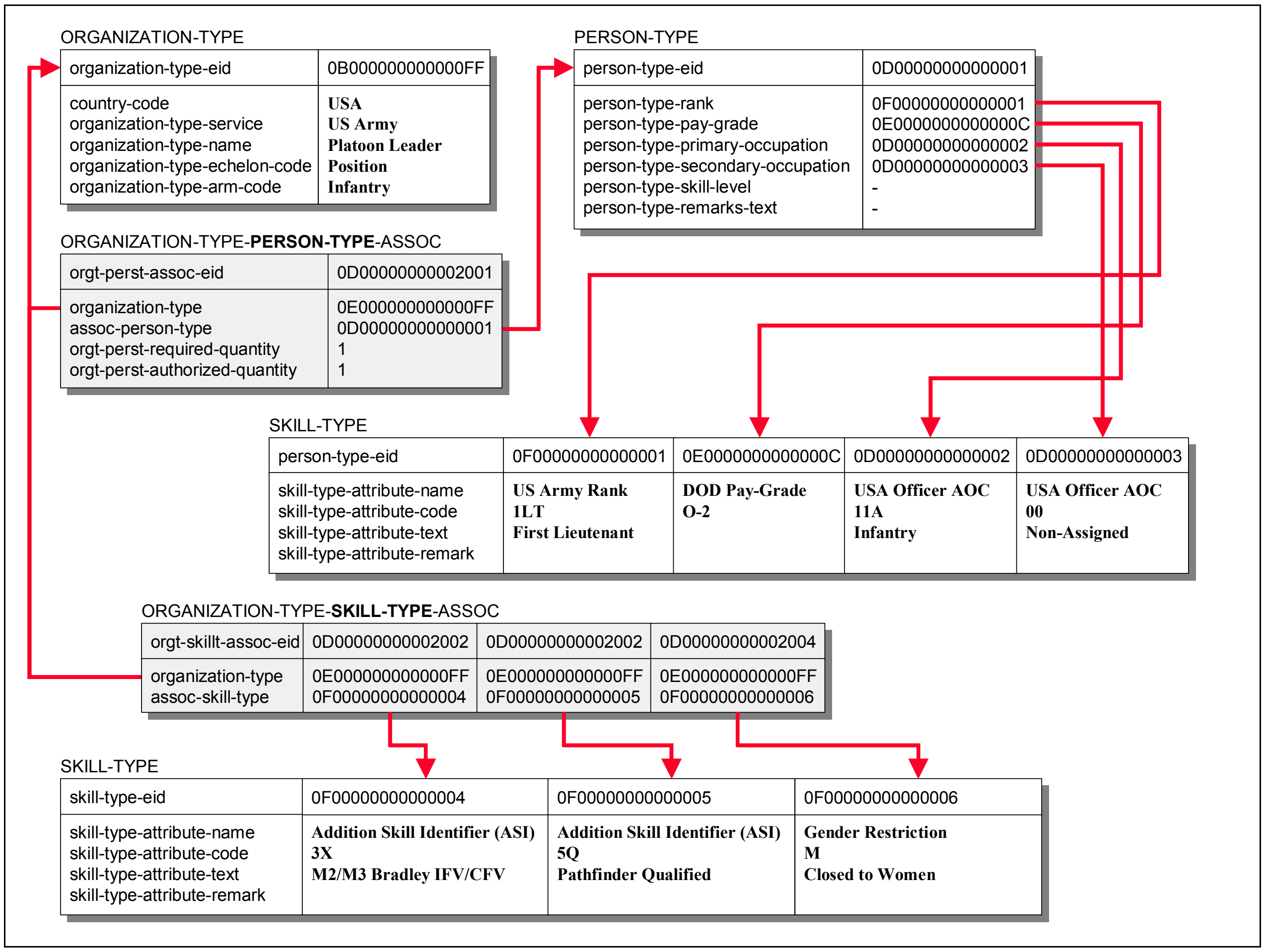

ORGANIZATION-TYPE
organization-type-eid 0B000000000000FF
country-code USA
organization-type-service US Army
organization-type-name Platoon Leader
organization-type-echelon-code Position
organization-type-arm-code Infantry
ORGANIZATION-TYPE-PERSON-TYPE-ASSOC
orgt-perst-assoc-eid 0D00000000002001
organization-type 0E000000000000FF
assoc-person-type 0D00000000000001
orgt-perst-required-quantity 1
orgt-perst-authorized-quantity 1
PERSON-TYPE
person-type-eid 0D00000000000001
person-type-rank 0F00000000000001
person-type-pay-grade 0E0000000000000C
person-type-primary-occupation 0D00000000000002
person-type-secondary-occupation 0D00000000000003
person-type-skill-level -
person-type-remarks-text -
SKILL-TYPE
person-type-eid 0F00000000000001 0E0000000000000C 0D00000000000002 0D00000000000003
skill-type-attribute-name US Army Rank DOD Pay-Grade USA Officer AOC USA Officer AOC
skill-type-attribute-code 1LT O-2 11A 00
skill-type-attribute-text First Lieutenant Infantry Non-Assigned
skill-type-attribute-remark
ORGANIZATION-TYPE-SKILL-TYPE-ASSOC
orgt-skillt-assoc-eid 0D00000000002002 0D00000000002002 0D00000000002004
organization-type 0E000000000000FF 0E000000000000FF 0E000000000000FF
assoc-skill-type 0F00000000000004 0F00000000000005 0F00000000000006
SKILL-TYPE
skill-type-eid 0F00000000000004 0F00000000000005 0F00000000000006
skill-type-attribute-name Addition Skill Identifier (ASI) Addition Skill Identifier (ASI) Gender Restriction
skill-type-attribute-code 3X 5Q M
skill-type-attribute-text M2/M3 Bradley IFV/CFV Pathfinder Qualified Closed to Women
skill-type-attribute-remark

Figure 27: Bundled Skill-Types

lower resolution, aggregate information about PERST data often called a "roll-up." An example might be the number of E-6/11M30 infantrymen that are in a rifle company.

In Figure 26 and Figure 27, the ORGANIZATION-TYPE-PERSON-TYPE-ASSOC table includes two quantity attributes, one for a *required* quantity and one for an *authorized* quantity.[58] Thus far, the ORGT has always been a position, so the quantity has been one. However, one could maintain the personnel requirement data at any resolution, and this is required in several battle management system data models.

In Figure 28, a US Army rifle platoon is selected as the echelon level of ORGT resolution and the associated person-type information is listed for this case. The platoon is composed of 32 soldiers (and their equipment) and reflects eleven different categories of requirements; that is, eleven different PERST/SKILLT combinations.

[58] This reflects the Army approach to force structure documentation. Although the quantity authorized usually equals the quantity required, this does have to be the case due to fiscal considerations.

```
Rifle Platoon (an ORGT):
 Line    #   PERST / SKILLT
 [ 1]    1 x O-2 11A00/3X,5R  where 3X = BIFV; 5R = Ranger
 [ 2]    1 x E-7 11M40/F7     where F7 = Pathfinder
 [ 3]    2 x E-6 11M30/J3     where J3 = BIFV Master Gunner
 [ 4]    1 x E-6 11M30
 [ 5]    1 x E-6 11M3G
 [ 6]    2 x E-5 11M2G
 [ 7]    4 x E-5 11M2O
 [ 8]   15 x E-4 11M1O
 [ 9]    1 x E-3 11M1O
 [10]    3 x E-3 11M1O/C2:    where C2 = Dragon Gunner
 [11]    1 x E-3 11M1O/B4:    where B4 = Sniper
```

Figure 28: Personnel Data Maintained at Resolutions Lower than Position

In this example, the eighth line indicates that there is a requirement for fifteen soldiers of grade E-4 with the qualification code of 11M10. There is no indication of the positions within the platoon that the people with these qualifications occupy. This is an example of aggregated PERST requirements at the platoon level of detail.

The capability to describe personnel (and materiel) requirements at any resolution has been provided via the concept of "establishment" that was originally defined in the Generic Hub (GH) data model (see footnote 18, pg 11). Establishments allow composition information about ORGTs to be defined at any resolution when the attributes are bundled (e.g., using PERST). Figure 29 illustrates the ORGT establishment view from the LC2IEDM, a derivative of the GH that has been modified to use EIDs. An ORGT may have several establishments (i.e., variants), each with a set of "detail" entities that define the quantity and type of subordinate organizations, equipment (i.e., materiel-type), and person-types. In this model, PERSON-TYPE has only two bundled attributes: "person-type-category-code" and "person-type-rank-code."

If the LC2IEDM were used to represent the information in Figure 28, each of the eleven lines would be implemented as a person-type[59] entity with a corresponding ORGANIZATION-TYPE-ESTABLISHMENT-PERSON-TYPE-DETAIL entity that contains a quantity value and the means to associate the PERSON-TYPE with the ORGANIZATION-TYPE-ESTABLISHMENT entity representing the "Rifle Platoon."

This configuration presents problems for both the generic and bundled attributes schemes presented thus far. The generic attribute scheme (Figure 24 and Figure 25) offers complete flexibility, but only allows the case when the ORGT is a position. There are no quantifiers in the association tables because each PERST (or SKILLT) represents a single qualifying characteristic and there is no way to assemble them into groups.[60] In these cases, the only valid quantifier is

[59] Actually, to be consistent with this report, the person-type would have to include skill-type information.

[60] Recall that the reason for breaking the attributes into individual characteristics is to avoid a combinatoric affect. One can always assemble all possible, required combinations of generic attributes into individual PERSTs and accept the data management challenge of tracking all the different combinations of attributes.

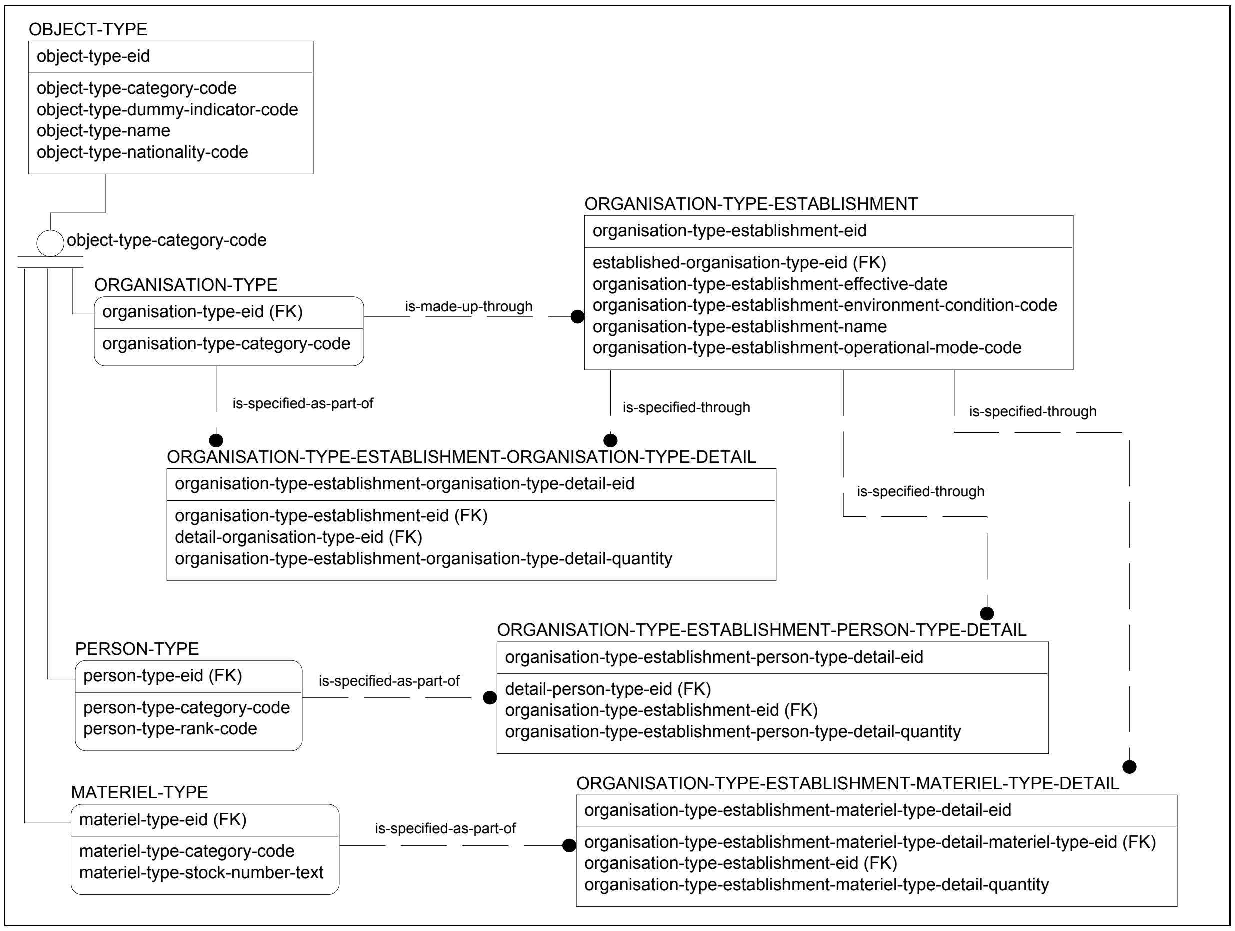

OBJECT-TYPE
object-type-eid
object-type-category-code
object-type-dummy-indicator-code
object-type-name
object-type-nationality-code
object-type-category-code
ORGANISATION-TYPE
organisation-type-eid (FK)
organisation-type-category-code
is-made-up-through
ORGANISATION-TYPE-ESTABLISHMENT
organisation-type-establishment-eid
established-organisation-type-eid (FK)
organisation-type-establishment-effective-date
organisation-type-establishment-environment-condition-code
organisation-type-establishment-name
organisation-type-establishment-operational-mode-code
is-specified-as-part-of
is-specified-through
is-specified-through
is-specified-through
ORGANISATION-TYPE-ESTABLISHMENT-ORGANISATION-TYPE-DETAIL
organisation-type-establishment-organisation-type-detail-eid
organisation-type-establishment-eid (FK)
detail-organisation-type-eid (FK)
organisation-type-establishment-organisation-type-detail-quantity
PERSON-TYPE
person-type-eid (FK)
person-type-category-code
person-type-rank-code
is-specified-as-part-of
ORGANISATION-TYPE-ESTABLISHMENT-PERSON-TYPE-DETAIL
organisation-type-establishment-person-type-detail-eid
detail-person-type-eid (FK)
organisation-type-establishment-eid (FK)
organisation-type-establishment-person-type-detail-quantity
MATERIEL-TYPE
materiel-type-eid (FK)
materiel-type-category-code
materiel-type-stock-number-text
is-specified-as-part-of
ORGANISATION-TYPE-ESTABLISHMENT-MATERIEL-TYPE-DETAIL
organisation-type-establishment-materiel-type-detail-eid
organisation-type-establishment-materiel-type-detail-materiel-type-eid (FK)
organisation-type-establishment-eid (FK)
organisation-type-establishment-materiel-type-detail-quantity

Figure 29: LC2IEDM Establishment (Modified for EIDs)

one. As a result, when using generic attributes there is no way to represent aggregate personnel information. In Figure 28, lines 9-11 would each be represented by the same PERST, the bundle of attributes indicating a soldier of grade E-3 with MOS 11M10. However, because of the different ASIs (e.g., C2, B3, etc.) these are three different qualifications.

One apparent solution is to associate a PERST entity with the optional SKILLT entities. However, this logic is incorrect because any association would indicate a universal relationship thus linking together all instances of these two entities; for example, the 11M1O MOS and the C2 ASI. Once this association is established, there is no way to distinguish between an 11M1O and an 11M1O/C2 entity. To achieve distinction, one must create variants of the 11M10 PERST, which causes a combinatoric expansion and defeats the original purpose of the limited bundling of attributes. So this technique does not offer an advantage.

However, there is a solution to this aggregation problem without submitting to the unacceptable combinatorial ramifications caused by building PERST entities for all required combinations of optional personnel attributes. The aggregation problem is caused by the choice to optimally manage personnel attributes at the individual level rather than at the aggregate (upper echelon) level. This choice is made because the individual level is exceedingly the most abundant (more than 80% of the ORG and ORGT entities will refer to positions and billets) and it is highly advantageous to be able to manage resources at the individual person level. Therefore, the aggregation of personnel attributes must be handled as *derived data*. Aggregation can be achieved by first grouping the position level ORGT data, and then filtering the associated PERST attributes. In other words, one executes aggregation by rolling up the ORGTs, not the PERSTs. This approach is illustrated in Figure 30, which mimics the technique used in the Army MTOE for displaying personnel information.

```
Rifle Platoon (ORGT):
 Line    ORGT                          PERST     /SKILLT
 [ 1]    1 x Platoon Leader          ⇨ O-2 11A00 /
 [ 2]                                ⇨           /3X
 [ 3]                                ⇨           /5R
 [ 4]    1 x Platoon Sergeant        ⇨ E-7 11M40 /
 [ 5]                                ⇨           /F7
 [ 6]    2 x Section Leader          ⇨ E-6 11M30 /
 [ 7]                                ⇨           /J3
 [ 8]    1 x Squad Leader            ⇨ E-6 11M3O
 [ 9]    1 x Squad Leader            ⇨ E-6 11M3G
 [10]    2 x Senior Gunners          ⇨ E-5 11M2O
 [11]    2 x Team Leader             ⇨ E-5 11M2O
 [12]    2 x Team Leader             ⇨ E-5 11M2G
 [13]    3 x Gunners                 ⇨ E-4 11M1O
 [14]    4 x BIFV Drivers            ⇨ E-4 11M1O
 [15]    6 x Automatic Rifleman      ⇨ E-4 11M1O
 [16]    2 x Grenadier               ⇨ E-4 11M1O
 [17]    1 x Radiotelephone Operator ⇨ E-3 11M1O
 [18]    3 x AntiArmor Specialists   ⇨ E-3 11M1O
 [19]                                ⇨           /C2
 [20]    1 x Rifleman                ⇨ E-3 11M1O
 [21]                                ⇨           /B4
```

Figure 30: Aggregate Personnel Information Accessed Via ORGT Associations

In this example, the person-type information is collected by first amassing the ORGT positions that form the leaves of the ORGT tree rooted at the desired echelon of aggregation, in this case, the platoon. The left columns list the number and name of the positions that are subordinate to the platoon. To the right of the position are the personnel attributes associated with that position. In this example, the structure reflects the scheme of using bundled attributes for mandatory fields via PERST entities and generic attributes via SKILLT entities for optional fields (e.g., ASIs). The "⇨"symbol indicates an association that links an ORGT with a PERST or a SKILLT. For example, lines 1-3 of Figure 30 denote three associations with the Platoon Leader position; one with a PERST (O-2, 11A00) and two with SKILLTs (ASIs 3X and 5R). Once the ORGT rollup is completed, the PERST and SKILLT information can be filtered and collated if required.

Clearly, using this approach, aggregation can be achieved using either the generic or bundled attribute technique, provided that the ORGT data is originally maintained with position resolution. In Army MTOE vernacular, this means that POSCO data may be rolled-up for any echelon, but “line number” entities (i.e., positions) must exist as the bottom level. Because all the service force structure documents use position data as the bottom level, the LC2IEDM “establishment” concept can be used, with minor adjustments, to represent personnel data, with either generic or bundled attributes, for all of the U.S. services.

This section has described three basic approaches to modeling personnel data: generalization hierarchies, generic attributes, and bundled attributes. Although apparently attractive, the development of a generalization hierarchy of specific personnel attributes will be difficult due to differences in perspective and the propensity for significant data consistency problems caused by combinatorial affects. Further, because a modification requires the changing of the data model (i.e., schema), it will add further complexity to the task of transition from disparate systems. Either generic or bundled attributes will work with several existing schemes.

Making the choice between generic or bundled attributes will depend more on familiarity issues than on hard technical arguments. If flexibility is a primary concern, then generic attributes are favored. If convenient groupings are preferred, then bundled attributes are advantageous. Both approaches have advantages and disadvantages due to their inherent characteristics. One option is to use both: bundled attributes for common, recurring (e.g., mandatory) attributes, and generic attributes for sparse, specialized (e.g., optional) attributes. Another option is to represent all personnel qualifications and skills, and then bundle the skills based upon common categories to improve understanding between different organizations. The authors prefer this last option. However, the ultimate choice will have to be decided by those who will use and provide the data.

5. RECOMMENDATIONS AND SUMMARY.

The purpose of this study was to investigate ways in which enterprise identifiers (EID) can be exploited in personnel systems to support interoperability between administrative and battle command systems. To accomplish this task, data modeling techniques were defined that:

- exploit the capabilities of EIDs,
- facilitate the integration of personnel data between administrative and battle command systems,
- apply across service and coalition boundaries, and
- are usable within an integrated Army Organization Server (AOS).

5.1 ***Recommendation 1:*** *Incorporate EIDs into personnel automated information systems as alternate keys.*

EIDs are surrogate keys that are unique across the enterprise. A surrogate key is a single attribute candidate key that contains no embedded intelligence (i.e., nothing can be gleaned from the key about the item being identified.). From a relational database perspective, EIDs are simply a single column in a database table that uniquely identifies a row of a table from every other row of every other table in the enterprise. There are several performance and maintenance advantages for using EIDs as primary keys, but all the interoperability advantages are gained when they are used as alternate keys. This allows legacy systems to retain all their current key structure, and still exploit the advantages of EIDs.

It is likely that any administrative personnel AIS will be based upon relational database technology. To implement EIDs as alternate keys, each table requires one additional column to hold the EID. All the existing columns remain unchanged. Consequently, the legacy applications interfaces (i.e., SQL interactions) remain unchanged. Initially, every row of every table will require an EID, but this is a simple process. After that, each time a new data item is inserted, it also receives an EID. The implementation technique to accomplish this is uncomplicated and uses a simple program called an EID server. Recommendation X explains this approach. The system managers determine the rate of exploitation of EIDs. This can be never, or immediately. There is no internal requirement to use EIDs; however, as their advantages are discovered, it is expected that they will rapidly be included into system updates.

5.2 **Recommendation 2:** *To facilitate interoperability, include a special external function called "Fetch-EID" into the interfaces of personnel AIS's along with support for XML*[61]**.**

For relational databases, the addition of a single table can dramatically increase interoperability. This table contains all the EIDs used in the database and the corresponding table in which the EID is an alternate key. When provided an EID, one can implement a function, called Fetch-EID, that executes a simple query that returns all the attributes of the row in which the EID is an alternate key. The attribute fields can be converted into XML to be returned as the arguments of the "Fetch-EID" function. This allows database implementers to obtain specific (usually well-known) authoritative data knowing only the EID of the required item, without having to know the schema in which the data is maintained.

5.3 **Recommendation 3:** *Implement EIDs as 64-bit sequences, or alternatively, as 16 characters using hexadecimal representation, building them as the concatenation of a global EID prefix and a locally generated EID suffix***.**

The difficult part of implementing an EID scheme is ensuring that there is no performance degradation while guaranteeing uniqueness. A technique to accomplish this is to create EIDs by concatenating two unique values, an EID prefix and an EID suffix. The prefix is a globally unique value, called an EID seed that is obtained from a centralized source. In this case, the source is the Army EID seed server (ESS) that is sanctioned by the Army CIO / G-6 and located at https://ess.arl.army.mil/. System managers obtain an ESS account and may then obtain 32-bit EID seeds. Once a seed is obtained, an EID server (ES) can be established that creates 64-bit EIDs by concatenating a locally generated 32-bit suffix to the 32-bit EID seed.

Note that this technique works for any sized prefix and suffix. The 64-bit value was selected is because it was determined to be the smallest value that would accomplish the task. EIDs must work in all environments, to include the low bandwidth communication conditions found in the wireless, tactical environment. For this reason, identifying the smallest possible value was determined to be a critical requirement. One way to think of the 64-bit value is that it allows 4.3 billion ES to be established that each produce 4.3 billion EIDs. This is a very large number, especially considering that EID seeds are obtained first-come, first-served so that there is no waste.

System managers can obtain as many seeds as they require. This number is normally determined by the system architecture and performance constraints. For example, if a project manager needs

[61] XML: The Extensible Mark-up Language. See footnote 28.

to deploy 5,000 independent, distributed systems, then 5,000 EID seeds can be obtained (one per system). Recall that the purpose of EIDs is to ensure uniqueness in a common format, nothing more. This approach guarantees that no matter where data tagged with an EID propagates, it will never collide with another EID within the enterprise. Further, the enterprise is determined by the set of ESS subscribers and is not restricted other than by restricting membership. Subscribers can be from any service, country, or type of organization as deemed necessary to conduct the business of the Department of Defense.

In summary, the EIDs in this scheme maintain three important characteristics:

- All EIDs are surrogate keys (fit in a single attribute, no embedded intelligence),
- All EIDs have the same size and form (in this implementation 64-bits),
- The size is chosen to be as small as possible (to facilitate limited bandwidth).

*5.4 **Recommendation 4:** Establish an initial interface between administrative and battle command systems for personnel data via the Army Organization Server (AOS).*

Force structure data forms the core of the information within battle command systems like the Army Battle Command System (ABCS). The US Army Force Management Support Agency (USAFMSA) maintains force structure data and is developing a new AIS called the Force management System (FMS). The FMS operational requirements document (ORD) calls for an Army organization server (AOS) as one of the products of FMS. The AOS will provide the equivalent of digitized MTOE information for tactical, battle command systems. It will be a database that provides hierarchical force structure data about both real organizations (ORG) and the templates from which they are constructed, called organization-types (ORGT).

For each type of organization, the ORGT data will include information about the type of materiel and people that are authorized, just like an MTOE. This data is named materiel-type (MATT) and person-type (PERST) data, respectively. The AOS will extend the hierarchical organization structure down to the individual soldier level. Thus, real organizations will extend down to the billet level, and each billet (ORG) will have a corresponding template (ORGT) called a position. When this occurs, about 70% of all the organizations in the AOS will be billets (or their corresponding positions). Every position will have associated PERST data that describes the requirements of the person that occupies the position. In Army vernacular, this includes occupational categories, grade and rank constraints, additional skills, and any other PERST information relevant to exercising battle command.

The AOS will be the authoritative source for digitized force structure related information. Army battle command systems will obtain their force structure data from the AOS, to include the personnel requirements data associated with each position. Therefore, this is an excellent point at which to inject administratively maintained personnel requirements data into tactical system. It is emphasized that the AOS only contains data related to the requirements for a person to fill a position, and not information about people. Consequently, there are no privacy constraints for this data.

5.5 **Recommendation 5:** *Make PERSCOM the authoritative source for Army person-type data maintained in the AOS, and develop automated procedures for loading the EID values into legacy systems planning to line up their data with the authoritative source.*

One of the basic tenets on which the AOS is designed is that the experts who develop the data should maintain it. Currently, USAFMSA receives personnel requirements information from PERSCOM via a variety of documents. USAFMSA then inserts that information into their databases for use in building force structure documents. The AOS will serve as a central access point for force structure data for most of the Army; therefore, it makes sense that PERSCOM maintain the PERST data directly rather than through a sequence of manual approaches.

However, the ease with which the EID-based key management will be adopted by the Services will in great measure depend on how burdensome the task of retagging legacy data will be. To avoid a situation where every participant begins to load EID values without any coordination with the rest of the members of the enterprise, thereby creating yet another type of stove-piping, namely, *identifier stove-piping*, adequate automated procedures should be developed that conduct the semantic matching between the records of the authoritative data source and the records of the target tables in the legacy systems. For a detailed analysis of this aspect of the EID implementation and adoption see Annex A below.

The scope of PERST data should include officer, warrant officer, and enlisted soldiers. Like the current MTOE, PERST attributes would cover occupational categories (e.g., AOC and MOS) and additional skills (e.g., SQI, and ASI). Further, both grade and rank can be included; for example, an E-8 First Sergeant can be designated differently from and E-8 Master Sergeant (this is especially helpful across military service boundaries). The attributes of the PERST entity can be extended to include specialized information; for example, a remark to indicate that a person must also serve as a radiotelephone operator while assigned to a position. Eventually, the AOS may include TDA force structure data at which time civilian requirements must also be added.

The most difficult part of this task is to develop the set of update rules to ensure that table maintenance occurs in a rigorous manner. One of the reasons for using EIDs is the ease by which relationship between data can be maintained. It is always more difficult to develop (and agree upon) a scheme by which multiple organizations can maintain a database. But the long-term advantages of having each expert maintain their portion of the data out-weigh the extra work, delay, and complications encountered at the beginning. USAFMSA and PERSCOM will have to work together to develop the procedures required to formally preserve a clean, efficient, and unambiguous force structure database.

5.6 **Recommendation 6:** *To facilitate inter-service exchange, model personnel data in the AOS using both bundled and generic attributes similar to a meta-model. The bundled skill-type option is recommended.*

Of the five basic battlefield entities, none is handled in a more diverse manner than PERSON-TYPE (PERST) data. PERST data is overloaded in all the services, meaning that it is used for more than one purpose. The same set of PERST attributes are used to describe both *the requirements for a position* (an ORGT) and *the qualifications of a person*. It is emphasized that PERST data describes the experiences and qualifications of people, and not the human beings themselves (i.e., it is not a PERSON entity which is a human being with attributes such as name, SSN, gender, race, or blood-type). The AOS will not contain any PERSON data, only PERST data, and only in reference to qualifying the requirements to fill a position.[62]

The challenging task is to select data structures to contain position requirement attributes that can be used across service (and national) boundaries even though each documents personnel qualities in very different ways. Unlike the organization and materiel domains, there will be very few attributes (or at least few enumerated ones) in the personnel domain that can be used across all the different systems.[63] A common theme discovered across the services is that personnel information includes both mandatory and optional elements. Mandatory elements consisted of rank and pay grade, primary and secondary occupational specialty, and skill level. Nearly all personnel data included some form of these attributes. There were a wide variety of optional attributes used sparsely within the domain. The ultimate objective is to provide a scheme that facilitates the exchange of PERST data across service and coalition boundaries when units are task organized into operating forces. To accomplish this, it is recommended that the mandatory/optional perspective be used in the scheme to categorize personnel information for use in battle command systems.

Three different strategies for modeling personnel data were presented: generalization hierarchies, generic attributes, and bundled attributes. Of these three, it is recommended that generic attributes be used to store atomic personnel attributes, but that a subset of these be bundled to improved information exchange between systems; this is illustrated in Figure 27, and is named *Bundled Skill-Types*. Because the PERSON-TYPE (PERST) name is already used in several data models, it is recommended that this name be used for the set of bundled, common attributes. An excellent entity name for the atomic personnel attributes is Skill-Type (SKILLT). This allows all personnel attributes to be maintained in a single, generic SKILLT table, each with an EID. The PERST table refers to the SKILLT table to identify the attributes associated with common categories of rank and pay grade, primary and secondary occupational specialty, and skill level. There will be one PERST entity for each position level ORGT entity, but there may be any number of direct links between the ORGT and SKILLT entities.

This combination of techniques is recommended because it offers several advantages. First, generic attributes offer flexibility to accommodate a wide variety of users. It divides the consistency task evenly between the data model and the application developer. Second, having the atomic personnel attributes located in a single table significantly eases the task of maintenance. Third, having a core set of common attributes bundled together via common themes allows disparate but corresponding entries in the SKILLT table to be identified. This enhances interoperability between the different users by providing a way to compare analogous schemes that use very different terms and configurations.

[62] Army PERST data is exemplified by the job titles and associated coding known as Military Occupational Specialties for enlisted and warrant officers and as Areas of Concentration for officers. Footnote 55 (on pg 35) provides URLs for detailed information about PERST data for all ranks in the Army, Navy, and Air Force.

[63] One of the few enumerated attribute that is usable across U.S. systems is "pay grade," but this is a unique case.

A recommended set of entities and attributes to accomplish this objective is provides in Figure 27. They are:

PERSON-TYPE:

person-type-rank
person-type-pay-grade
person-type-primary-occupation
person-type-secondary-occupation
person-type-skill-level
person-type-remarks-text

SKILL-TYPE:

skill-type-attribute-name
skill-type-attribute-code
skill-type-attribute-text
skill-type-attribute-remark

A person-type-name attribute might also be included, but its usefulness is dubious. There are many options and alternative ways to define these entities. The key is finding the right mix of capabilities that provides a simple and realistically maintainable solution. Compromise is often the means to success.

ANNEX A

ANALYSIS OF OPTIONS FOR EID IMPLEMENTATIONS

PREFACE

This Annex addresses in detail a series of alternatives for the adoption and implementation of database key-management schemes based on globally unique enterprise identifiers (EIDs). Use of such a *record naming convention* in conjunction with appropriate DoD policies and procedures can provide a readily achievable degree of enterprise-wide data interoperability—one of the key components of the *infostructure* needed to support Network-Centric Warfare solutions. Adoption of EIDs can facilitate the creation of agency, Service, contractor, technology and functional area independent information services when addressing, for example, military units, logistics, and personnel data in the various automated information systems that support the warfighter.

As recommended by the sponsor, the approach takes into consideration the experience and results already obtained by the Army in previous studies in the areas of force management, and logistics, and seeks to leverage them for a comprehensive formulation.

Table Of Contents

Preface 58

Summary 62

I. Basic Concepts And Definitions 67

A. Definition of Entities, Attributes and Relationships 67

B. Data Modeling Notation 69

II. Technical Alternatives Analyses 71

A. Technical Alternatives for the EID Structure 71

B. Technical Alternatives for the Use of EIDs in Existing Systems 73

C. Technical Alternatives for the Use of EIDs in New Systems 84

D. Performance Considerations When Using ORG IDs in New Systems 105

E. Considerations on Loading EID Values into Legacy Systems 108

List Of Figures

Figure A-1. Abstraction Process from Enterprise Objects to Data Entities and Their Attribution 67

Figure A-2. Connections among Data Entities: Relationships 68

Figure A-3. Generalizations and Resulting Taxonomy 69

Figure A-4. Summary of the IDEF1X Notation 70

Figure A-5. First Technical Alternative for Globally Unique EIDs, e.g., ORG ID 71

Figure A-6. Second Technical Alternative for Globally Unique EIDs, e.g., ORG ID 72

Figure A-7. Partial Insertion of Globally Unique EIDs Within a Hierarchy, e.g., ORG ID In Legacy Systems 73

Figure A-8. Overall Insertion of Globally Unique EIDs Within a Hierarchy, e.g., ORG ID In Legacy Systems 74

Figure A-9. Partial Insertion of Globally Unique EIDs, e.g., ORG ID, Within Parent-Child Identifying Relationships In Legacy Systems 75

Figure A-10. Overall Insertion of Globally Unique EIDs, e.g., ORG ID, Within Parent-Child Identifying Relationships In Legacy Systems 76

Figure A-11. Overall Insertion of Globally Unique EIDs, e.g., ORG ID, Within Parent-Child Non-Identifying Relationships In Legacy Systems.. 77

Figure A-12. Partial Insertion of Globally Unique EIDs, e.g., ORG ID, Within Double-Associative Parent-Child Relationships In Legacy Systems 78

Figure A-13. Globally Unique EIDs, e.g., ORG ID, as Primary Keys for Generalizations In Legacy Systems 79

Figure A-14. Globally Unique EIDs, e.g., ORG ID, as Primary Keys for Identifying Parent-Child Relationships In Legacy Systems... 80

Figure A-15. Globally Unique EIDs, e.g., ORG ID, as Primary Keys for Associative Relationships In Legacy Systems 81

Figure A-16. Globally Unique EIDs, e.g., ORG ID, as Primary Keys for Double Associative Relationships In Legacy Systems.. 82

Figure A-17. Globally Unique EIDs, e.g., ORG ID, as Primary Keys for Non-Identifying Parent-Child Relationships In Legacy Systems 83

Figure A-18. Globally Unique EIDs, e.g., ORG ID, as Primary Keys with Generalizations Recast as Non-Identifying Z-Relationships... 85

Figure A-19. Globally Unique EIDs, e.g., ORG ID, as Primary Keys with Generalizations Recast as Identifying Z-Relationships 86

Figure A-20. Globally Unique EIDs, e.g., ORG ID, as Primary Keys with Identifying Relationships Recast as Non-Identifying-Relationships.. 87

Figure A-21. Globally Unique EIDs, e.g., ORG ID, as Primary Keys with Identifying Relationships in an Associative Entity Recast as Non-Identifying-Relationships 88

Figure A-22. Globally Unique EIDs, e.g., ORG ID, as Primary Keys and Recasting of all Relationships as Record-Level Entries within a Master Lookup Table 89

Figure A-23. Any Identifying Parent-Child Relationship Maps to a Record-Level Entry within the MasterLookup 90

Figure A-24. Any Identifying Parent-Child-GrandChild Relationship Maps to a Record-Level Entry within the MasterLookup 91

Figure A-25. Any Non-Identifying Parent-Child Relationship Maps to a Record-Level Entry within the MasterLookup 92

Figure A-26. Any Generalization of Any Depth Maps to a Record-Level Entry within the MasterLookup 93

Figure A-27. Any Associative Entity Maps to a Record-Level Entry within the MasterLookup 94

Figure A-28. Any Double Associative Entity Maps to a Record-Level Entry within the MasterLookup 96

Figure A-29. Alternative Approach for Handling a Double-Association via MasterLookup 97

Figure A-30. Mapping of LC2IEDM Holding using EID and MasterLookup 100

Figure A-31. Example of MasterLookup entries for Unit Holding 101

Figure A-32. Mapping of LC2IEDM Organization-Organization-Association using EID and MasterLookup 103

Figure A-33. Example of MasterLookup entries for Organizational Hierarchy 104

Figure A-34. Segmentation of the MasterLookup by Area as a Means to Prevent Performance Degradation 105

Figure A-35. Segmentation of the MasterLookup at Record-Level as a Means to Prevent Performance Degradation 106

Figure A-36. A Process Context for Migrating EIDs into Legacy Databases 110

Figure A-37. Procedure for Automating EIDs Insertion into Legacy Databases 113

Figure A-38. Steps for EIDs Insertion as Primary Keys into Legacy Databases 122

LIST OF TABLES

Table 1. Notional Legacy Database Table for Personnel Data 117

Table 2. Notional Authoritative Data Source Table for Person Data 117

Table 3. Definitions of the Functions, Subroutines, and Class Methods used in the “MigrateToChildren” and “MigrationDriver” Algorithms 120

SUMMARY

A. BACKGROUND

As indicated in a previous report prepared for J6I[64], *key management*[65] constitutes one of the four basic components for achieving *database data interoperability*[66]. It is important to realize that database data interoperability is a special case in the general area of *interoperability.* Other facets of data interoperability may be equally pertinent to the implementation of Network Centric Warfare solutions, e.g., the interoperability of various software applications[67] used in information systems within the DoD enterprise. Those areas of interoperability are, however, outside of the scope of the present analysis.

The purpose of this analysis, therefore, is to assess the technical alternatives for a DoD-wide globally unique EID, such as ORG ID.[68] There are two parts to the assessment. The first pertains to the structure of the EID itself. The second addresses its use as the primary record identifier—what is commonly referred as the "key" of a table in a relational database. The analysis addresses technical alternatives for both existing as well as planned systems.

The analysis also shows that the global uniqueness of the enterprise identifier (EID) offers some interesting possibilities for how to model data that are not possible with traditional key management schemes where uniqueness is only guaranteed at the table level by the Relational Database Management System (RDBMS) engine.[69]

---

[64] *Org Id Requirements*, IDA, Unclassified, 2002.

[65] **Key Management:** the procedures for the generation, distribution and maintenance of the values used within the tables of a relational database to uniquely identify the records in said tables.

[66] **Database Data Interoperability:** The ability to reuse structured data among information systems implemented with commercial RDBMs. It requires both semantic and syntactic overlap for the data targeted for exchange. In other words, not only must the object be understood in the same way by both systems, but it must be characterized also in the same way. Ideally, both systems share the same metadata for the objects, and hence can reuse their data. However, it is unlikely that every RDBMS implementation will conform to the same data model, and, therefore, it is better to agree to a common information exchange data model (IEDM). Users are free to implement their databases according to their needs and resources, but must transform their data prior to exchange in order to participate. Data reuse is guaranteed for the objects captured in the IEDM.

[67] **Software Application Interoperability:** The ability of software applications to manipulate non-structured data, e.g., binary large objects (BLOB) and character large objects (CLOB). This type of interoperability is mostly achieved via implementation of common formats for data types such as BLOB. Technical Architectures such as the Army Joint Technical Architecture (A-JTA) provide the foundation for software application interoperability.

[68] For purposes of this paper we will refer to the globally unique enterprise identifier (EID) for military units as the ORG ID.

[69] Since every record within a table of an RDBMS that uses EIDs as keys is not only uniquely identifiable within that table but within the whole database, as well as within all the databases of the enterprise that implement such a key management, it is possible to create relationships directly at the record level rather than at the table level.

B. TECHNICAL ALTERNATIVES FOR THE EID STRUCTURE

Global uniqueness is the essential characteristic of the enterprise identifier for all relevant battlefield objects, namely, FACILITY, FEATURE, MATERIEL, ORGANIZATION and PERSON, as well as any other conceptual object required for their management, such as PLAN, MISSION, ACTION, DOCUMENT, etc. This can be achieved as follows:

- By concatenating a centrally managed globally unique prefix with a locally managed and locally unique suffix
- By collocating with each system within the enterprise an instance of an application written so that it can produce globally unique identifiers with infinitesimally small likelihood of creating duplicates

The first approach provides the users with the ability to query the prefix generator to find out information about the allocation of the prefixes to users and permits some level of control as to who may be authorized to create EIDs within the enterprise. As long as the correct procedures for the generation of local suffixes are followed, the keys generated under the first approach with a given prefix are guaranteed to be unique.

The second approach relies on the robustness of the code that generates the values of the EID. Under this approach, there is no way to find out who may be using a given EID, since its structure does not depend on a centrally managed prefix. If, on the one hand, the enterprise creates the EID generator then it may control the creation of EIDs by limiting who may have access to the code. If, on the other hand, the enterprise uses a commercial product, it may not be possible to control the generation of EIDs outside of the enterprise.

C. TECHNICAL ALTERNATIVES FOR THE USE OF EIDs

Commercial RDBMSs ensure local uniqueness of the primary identifier for each table. Such local keys are used to execute SQL queries, and to enforce referential constraints—child tables contain copies of the parent keys, so-called migrated keys, which permit cascade deletes and updates.

In the case of existing systems, the technical alternatives are:

- Use the EIDs as alternate keys while retaining the current key structure
- Use the EIDs as primary keys with retention of the current keys as alternate keys to support code written against them

In the case of new systems, the technical alternatives are:

- Retain the traditional structure of relational databases, i.e., identifying parent-child and supertype-subtype relationships. Use the EIDs as primary keys but only for the parent entities, i.e., allow for the migration of the parent key into the child table. Provide only sequential indexes for the child entities.
- Remove all identifying parent-child and supertype-subtype relationships. Assign EIDs to all entities. But retain non-identifying relationships among the entities.

- Remove all relationships among the entities. Assign EIDs to all the entities. Create all pertinent relationships by establishing record-level associations based on the EIDs of each table.

D. PERFORMANCE CONSIDERATIONS FOR THE USE OF EIDs

When using EIDs such as ORG ID for record identification in participating information systems within the DoD enterprise, one of the most important criteria is response time for operations dependent on access to records tagged with such identifiers.

Performance degradation may be due to the following:

- Structure of the EIDs themselves
- Choice of implementation as the record identifier within a particular RDBMS—either as primary key or as alternate key

With respect to the first possible impact of using EIDs as the record identifier the following alternatives can be explored:

- If performance degradation arises from lack of native SQL data types in the RDBMS to store the EID, then either indexing or hash tables may be appropriate.

For example, choosing EIDs as 64 bit long integers may not be supported by older RDBMSs such as Microsoft SQL 7.0. Desktop database applications, such as Microsoft Access, may record the EIDs using some other form of internal representation, e.g., the currency data type. If EIDs are implemented with commercial products, such as Microsoft database Replication Identifier (GUID) which is a 128 bit long integer, some databases may be able to record the EIDs only as the ASCII string corresponding to the Hexadecimal representation of the GUID values. In such cases, queries with large data sets may be less efficient than if the EIDs were natively stored as numeric values. Indexing or hash tables may be an alternative to ameliorate the problem of unacceptably long response times.

The second possible source of performance degradation is the use of EIDs as alternate keys or primary keys in conjunction with the regular structuring of relationships—identifying, non-identifying and supertype-subtype hierarchies. However, no special performance degradation should occur because the keys used are EIDs rather than keys unique at the table level only—i.e., the traditional RDBMS generated table identifiers. If no key migration is allowed, and every record in every table is uniquely identified via EIDs, then relationships can, for example, be captured at the record level rather than at the table level. One possibility for implementing this approach involves the creation of physical tables—called in this analysis *Master Lookup tables*—in which all the relationships are encoded via pairs of entries that correspond to, for example, a parent-child relationship. It becomes apparent that when the number of records in the database increases, the number of entries in a single Master Lookup table can become very large. Since SQL queries will require the traversing of the Master Lookup table, this may have a visible impact in the response time of a system implemented this way. When this occurs, then the following alternatives can be explored:

- segmentation by 'functional area' of the Master Lookup table data encoding the record-level relationships

In other words, instead of one large lookup table where the data for all the relationships among all the records of the database reside, multiple lookup tables may be used to capture information pertinent to C2 operations. A different lookup table also may be used for planning data or technical architecture data.

- segmentation by record of the Master Lookup table data encoding the record-level relationships.

Again, instead of implementing a single large lookup table to relate the EIDs of the records related in some form to each other, multiple lookup tables may contain the EIDs for the relationships segmented according to specific criteria. For example, if military units are related to other military units under operational command, and they are also related to each other under future reorganization plans, then the owner of the information system may maintain two different lookup tables. The information system interface will search one or the other lookup table based on the entry point into the system by a user or another external system. For example, if the system is set up to provide information via the web, e.g., as a web portal, then the system knows which interface is initiating the query. Also, code behind each web page can be tailored so that the system automatically searches the appropriate lookup table. Additional measures may include dynamic resorting of the lookup table to rank relationships based on the frequency with which they are used. For instance, they may be placed at the beginning of the lookup table rather than reside within it by entry or creation order.

E. CONSIDERATIONS ON LOADING EID VALUES INTO LEGACY SYSTEMS

Finally, even if the modifications to the structure of the legacy database can be accomplished in a relatively straightforward manner, it is as important to assess how the modified structure will load the pertinent EID values into the tables while retaining the legacy data and functionality present prior to the modifications. The present analyses show that the actual loading of EIDs in general does not represent a major obstacle.

However, allowing participating systems in the DoD enterprise to begin loading any kind of EID values is likely to create a different type of stove-piping, namely, *identifier stove-piping*. By this it is meant that the systems will all have globally unique EIDs but won't be able to use them as effectively for the purpose of data integration because of identifier stovepiping. In other words, even though every one of the enterprise systems has EIDs, two systems using the same type of reference data still won't be able to use their EIDs for data retrieval and updates UNLESS the EID values for commonly used data—so-called reference data—are taken from authoritative data sources.

Adopting this policy will require the automation of the procedures for (1) matching records in legacy systems with those of authoritative data sources that have already been tagged with EIDs; (2) retrieving the EID values assigned to the records of the authoritative data source; and then (3) populating the legacy system tables with them. Absent these automated procedures, legacy systems with large data sets may be reluctant or incapable of adopting EIDs, since the manpower requirements may not be supported within their fiscal constraints.

Fortunately, techniques from machine learning, such as Bayesian algorithms, appear to have sufficient robustness to ease the transition from local keys to EIDs aligned with those of authoritative data sources. As DoD moves towards a coherent approach for data management, it is recommended that these techniques be further matured and made part of the tool-kit for developers supporting the Services and agencies.

I. Basic Concepts And Definitions

A. DEFINITION OF ENTITIES, ATTRIBUTES AND RELATIONSHIPS

For the purpose of this analysis, the following concepts are defined as follows:

- ENTITY: a distinguishable object within the enterprise, such as person, place, event, inanimate thing, administrative construct, conceptualization, etc., about which information is kept
- ATTRIBUTE: a property or characterization of an entity
- RELATIONSHIP: a connection between two entities
- TAXONOMY: a particular model, or classification tree, about the nature and relations of the enterprise entities
- ONTOLOGY: the universe of objects to be considered within a given enterprise
- DATA MODEL: A specification of the ENTITIES, ATTRIBUTES and RELATIONSHIPS as well as the business rules needed to support the information requirements of a particular area of the enterprise

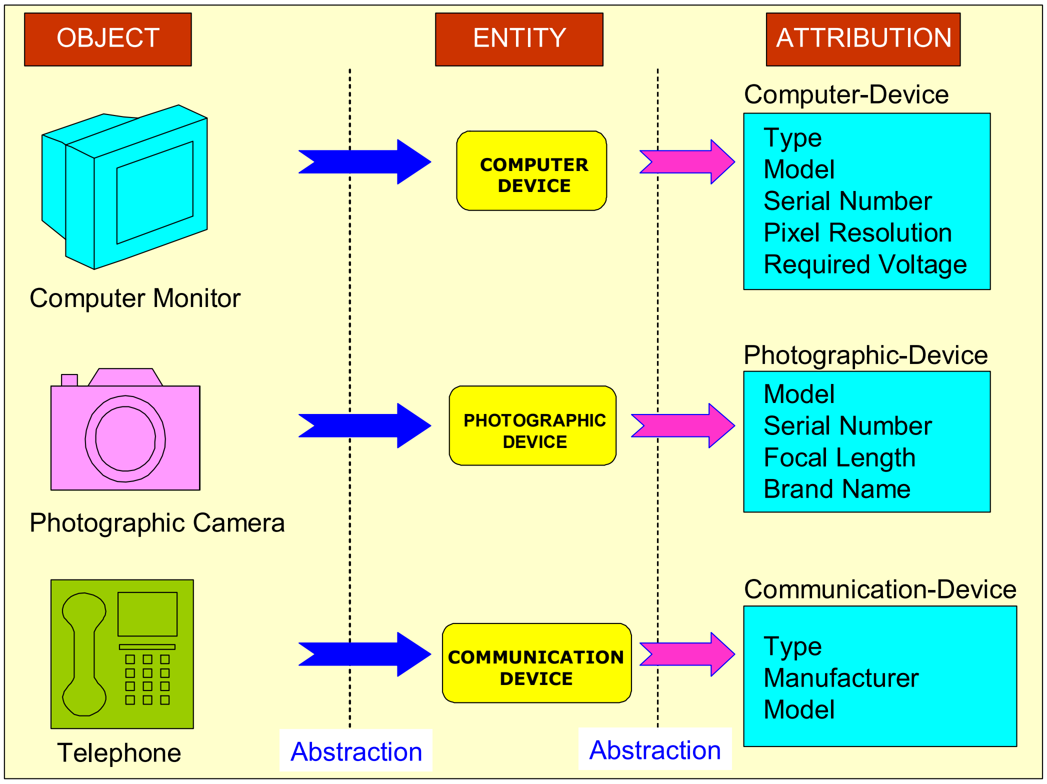


Figure A-1. Abstraction Process from Enterprise Objects to Data Entities and Their Attribution

Figure A-1 above shows the abstraction process that takes place when the enterprise objects are conceptualized as entities, and then, based on the requirements of the enterprise, how the entities are attributed.

Figure A-2 below depicts how data entities pertaining to a given enterprise are related to each other in terms of functions, as well as more abstract connections that are meaningful within the context of the enterprise business.

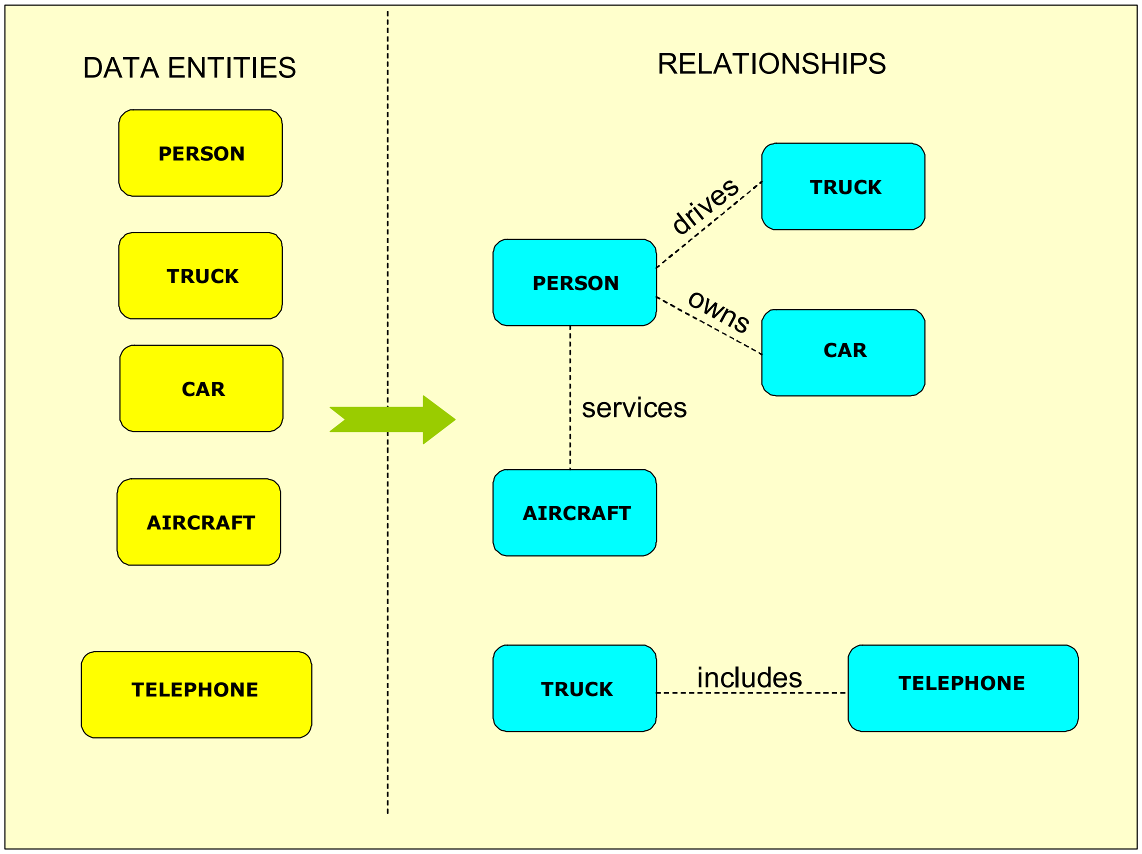


Figure A-2. Connections among Data Entities: Relationships

Once a particular enterprise determines the universe of its objects—its ontology, the corresponding data entities can be further abstracted via generalizations. Such hierarchical ordering creates specific taxonomies for the enterprise.

On the one hand, except for taxonomies built strictly on the basis of the actual sub-components of the data entities—an AIRCRAFT is composed of one AIRCRAFT-BODY, two AIRCRAFT-WINGs, one or more AIRCRAFT-ENGINEs, etc.,—most taxonomies based on use or functionality tend to be unstable over time. They also are likely to show inconsistencies across the different domains of the enterprise. On the other hand, highly abstract hierarchies can provide great conceptual stability. However, they are more difficult to grasp by the non-experts, and, therefore, are less likely to gain widespread acceptance in other communities of interest.

Figure A-3 below shows how specific data entities are abstracted through generalizations into more generic supertype entities. In this abstraction process one builds taxonomies for the particular domain of the enterprise for which these data entities have been postulated. A similar approach can be taken if, instead of starting with an ontology closely related to the real objects of the enterprise, one begins with an ontology based on the characteristics of the data itself. Consider, for instance, when all data entities are themselves instantiations of the class ENTITY, and all relations are themselves specific cases of the class RELATIONSHIP. The resulting construct is called a meta-model, and it is quite useful when developing tools to represent specific models dealing with the more concrete data entities.

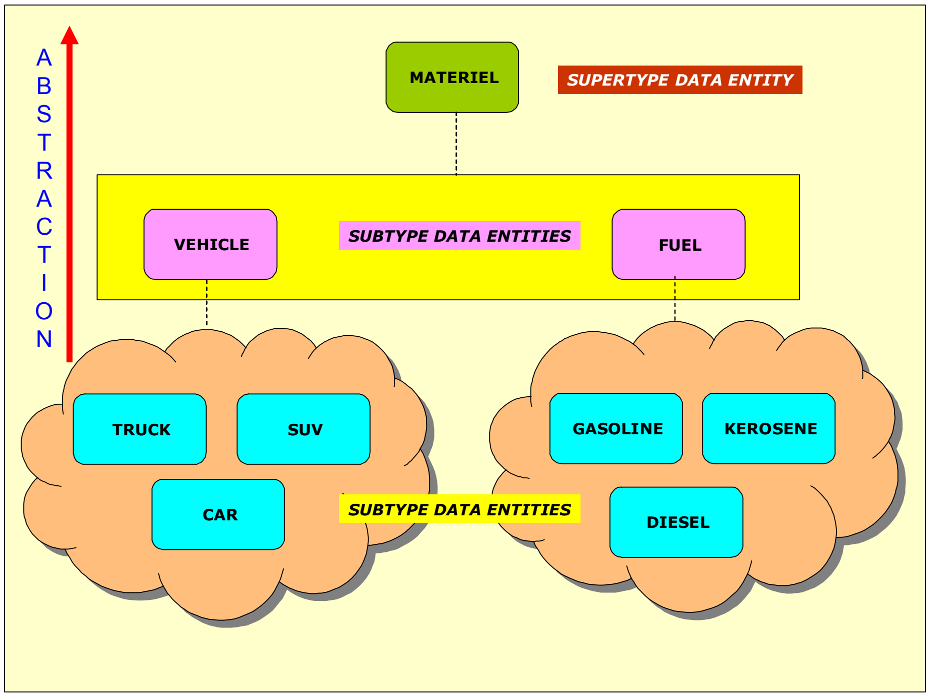


Figure A-3. Generalizations and Resulting Taxonomy

B. DATA MODELING NOTATION

The previous section presented a high-level description of the concepts and typical processes involved in the specification of information requirements for any given domain of an enterprise. These activities constitute information modeling, or, more colloquially, data modeling. Over the years, various methodologies for how to perform these tasks in a structured approach have been developed. Within DoD, the methodology known as IDEF1X (FIPS-184) was adopted as the standard representation, and it is the one that will be used in this analysis. Figure A-4 below summarizes the IDEF1X symbology.

The basic notation of IDEF1X is as follows:

- INDEPENDENT ENTITY: a data entity that does not depend on any other data entity for the identification of its records
- DEPENDENT ENTITY: a data entity that requires one or more data entities for the identification of its records
- PRIMARY KEY: the attribute or attributes that uniquely identify the records of an entity. If the primary key consists of a single attribute the primary key is said to be simple. Otherwise it is composite. Most independent entities contain simple primary keys. Dependent entities always have composite primary keys.

- CANDIDATE KEY: an attribute or group of attributes that might serve as the primary key of a data entity
- ALTERNATE KEY: A candidate key that has not been chosen as a primary key.
- FOREIGN KEY: the key of the parent entity that is contributed via a relationship to the child entity. If the foreign key by itself or in conjunction with other attributes serves to identify the records of the child entity then it is a PRIMARY FOREIGN KEY.
- IDENTIFYING RELATIONSHIP: a relationship in which all the attributes that constitute the primary key of the parent data entity become part of the primary key of the child entity.
- NON-IDENTIFYING RELATIONSHIP: a relationship where all the attributes that constitute the primary key of the parent data entity do not become part of the primary key of the child entity.

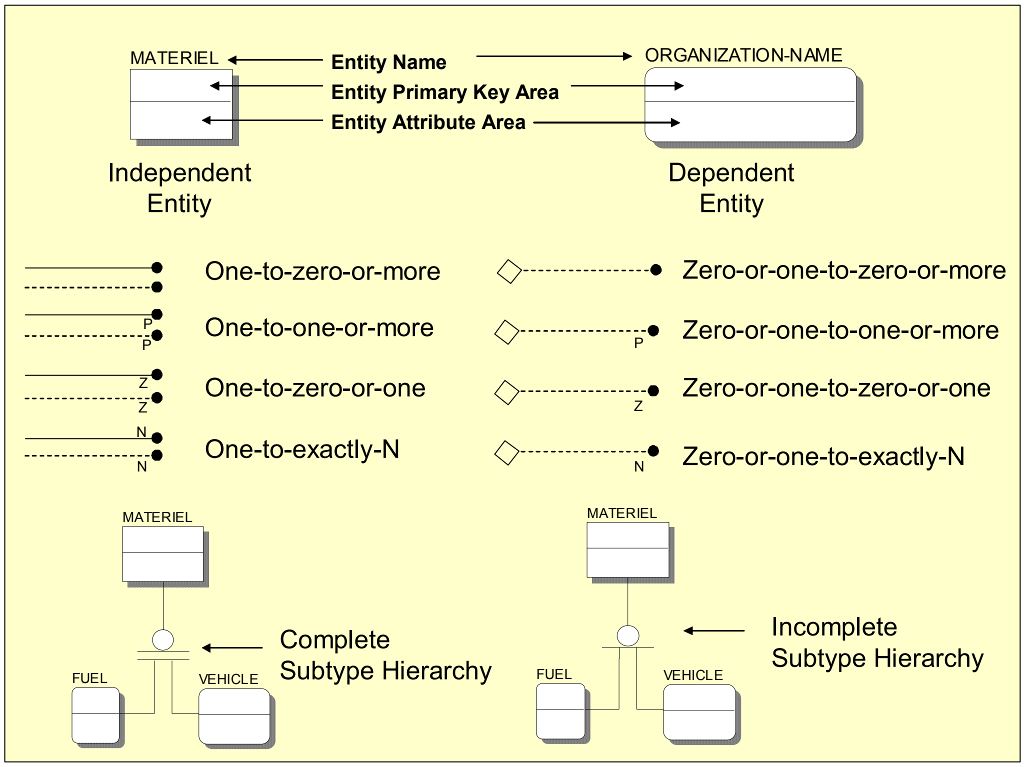


Figure A-4. Summary of the IDEF1X Notation

In IDEF1X, independent entities are represented as rectangles with square corners. Dependent entities are represented as rectangles with rounded edges. Identifying relationships are depicted as solid lines starting at the parent entity and terminating at the child entity. Non-identifying relationships use a doted line instead. A black dot at the child's end customarily represents the so-called "many" side of the relationship. In addition, relationships can specify a particular "cardinality", i.e., how many instances of the child entity are related to the parent entity.

II. Technical Alternative Analysis

A. TECHNICAL ALTERNATIVES FOR THE EID STRUCTURE

Global uniqueness is the essential characteristic of the enterprise identifier for all relevant battlefield objects, namely, FACILITY, FEATURE, MATERIEL, ORGANIZATION and PERSON, as well as any other conceptual object required for their management, such as PLAN, MISSION, ACTION, DOCUMENT, etc. Figure A-5 depicts the basic concept underlying the first technical alternative for achieving global uniqueness, namely, the concatenation of a globally unique prefix with a locally unique suffix.

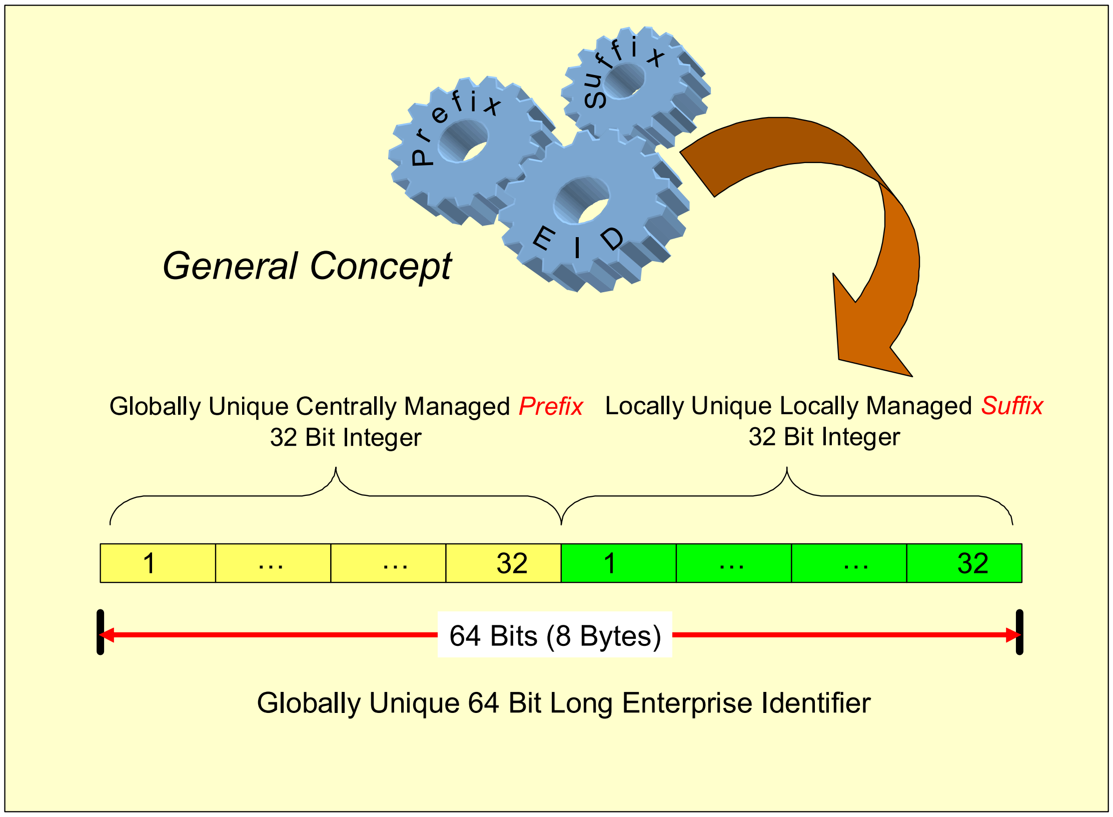

Prefix
Suffix
EID
General Concept
Globally Unique Centrally Managed Prefix
32 Bit Integer
Locally Unique Locally Managed Suffix
32 Bit Integer
1
…
…
32
1
…
…
32
64 Bits (8 Bytes)
Globally Unique 64 Bit Long Enterprise Identifier

Figure A-5. First Technical Alternative for Globally Unique EIDs, e.g., ORG ID

Specifically, such a technical alternative may be implemented by instituting a centralized management of the globally unique prefixes so that:

- the participating enterprise members can obtain them readily, and
- they can be assured that once a prefix has been released to a requestor, no one else within the enterprise will receive the same prefix.

In addition, the participating members must ensure that the locally managed suffixes are all unique—such as when an RDBMS creates a local key as auto-number, i.e., a sequentially increasing integer that never repeats. Figure A-5 shows how the Army has proposed to create such a type of EID, namely, by choosing 32 bit long integers for both the prefix and the suffix portions of the enterprise identifier. Within the SHAPE initiative called the Army Tactical C2 System (ATCCIS), a similar approach has been demonstrated where the prefixes are assigned to each participating nation, rather than to individual organizations.

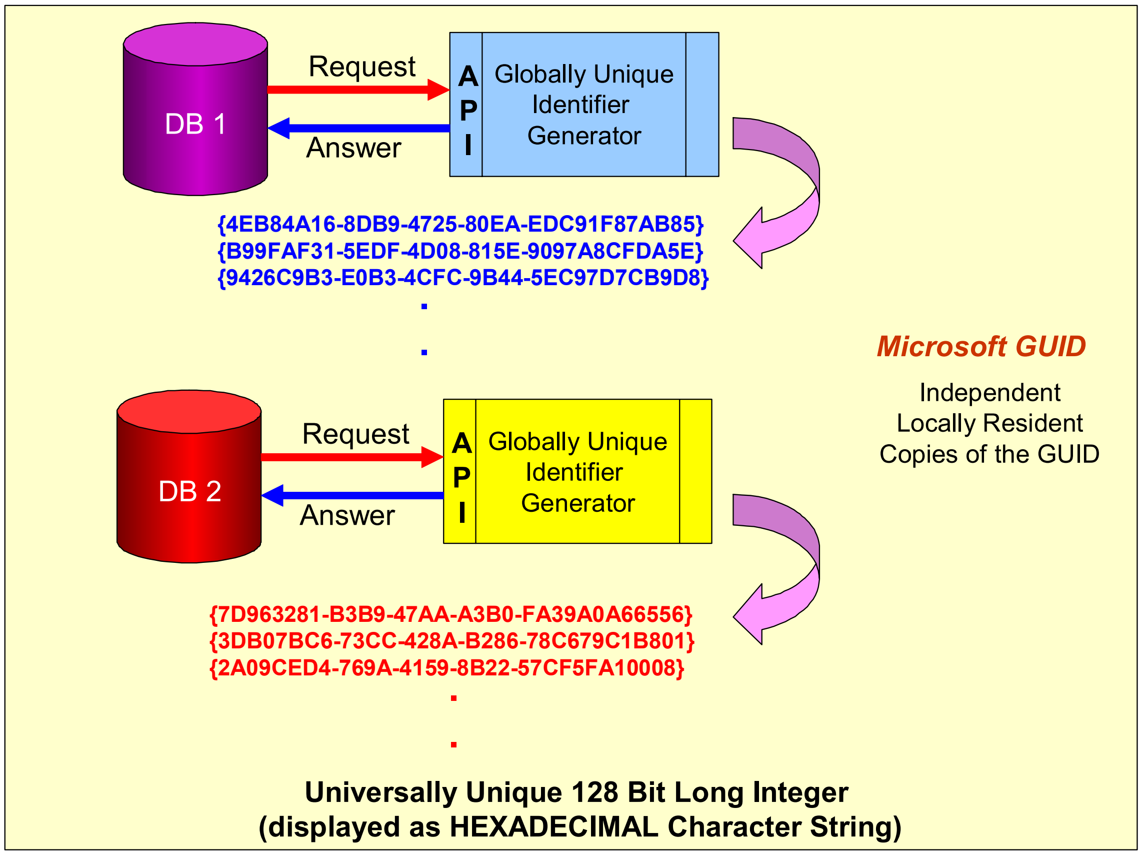

DB 1
Request
Answer
API
Globally Unique Identifier Generator
{4EB84A16-8DB9-4725-80EA-EDC91F87AB85}
{B99FAF31-5EDF-4D08-815E-9097A8CFDA5E}
{9426C9B3-E0B3-4CFC-9B44-5EC97D7CB9D8}
Microsoft GUID
Independent Locally Resident Copies of the GUID
DB 2
Request
Answer
API
Globally Unique Identifier Generator
{7D963281-B3B9-47AA-A3B0-FA39A0A66556}
{3DB07BC6-73CC-428A-B286-78C679C1B801}
{2A09CED4-769A-4159-8B22-57CF5FA10008}
Universally Unique 128 Bit Long Integer
(displayed as HEXADECIMAL Character String)

Figure A-6. Second Technical Alternative for Globally Unique EIDs, e.g., ORG ID

Figure A-6 depicts another alternative for structuring globally unique EIDs. Under this approach, independent instances of executable software objects with appropriate interfaces, i.e., APIs, are placed within each participating information system. Whenever a record in a table from an RDBMS, or an object in an object oriented system, is created, the application requests a new EID from the GUID generator.

The code within the GUID generator is written so that the likelihood of two instances of the generator creating the same value is extremely small. Such an approach has been implemented by Microsoft for their database Replication Identifier. Each table can have as its primary key such a replication identifier, and no two records will receive the same value. In the Microsoft implementation the Replication Identifier has a 128 Bit long integer internal representation, i.e., the RDBMS stores the values in that format. The user, however, sees the values as a character string corresponding to the hexadecimal representation of the 128 bit long integer value.

B. TECHNICAL ALTERNATIVES FOR THE USE OF EIDs IN EXISTING SYSTEMS

In the case of existing systems the main options are:

- use the EIDs as alternate keys while retaining the current key structure
- use the EIDs as primary keys with retention of the current keys as alternate keys to support code written against them

1. USE OF EIDs AS ALTERNATE KEYS IN EXISTING SYSTEMS

The first option basically implies that tables in databases of existing systems can begin to participate in data exchanges based on EIDs by simply adding a new attribute to them that can be used as an alternate key.

The impact to the databases is minimal since their key management is left untouched. In other words, all the code that has been written against the keys of each table will continue to operate since the EID appears to the RDBMS engine as another attribute below the line of the primary key area.

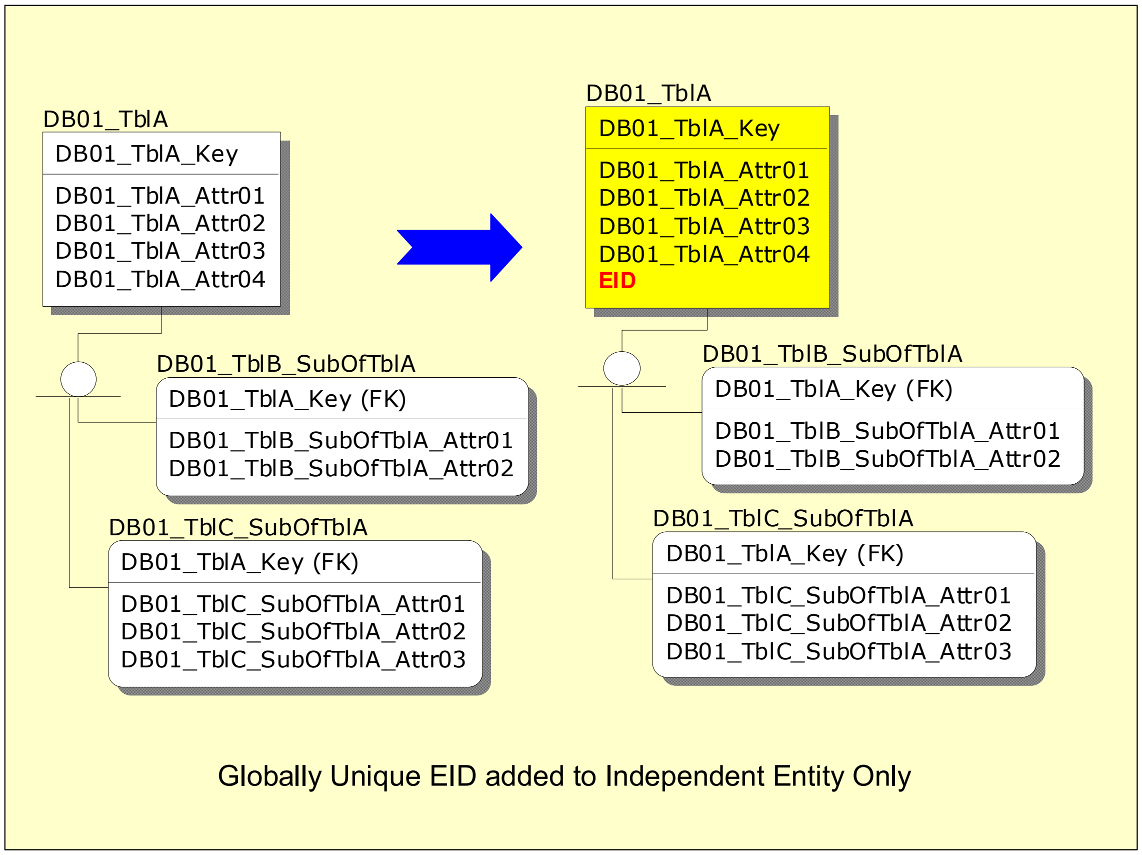


Figure A-7. Partial Insertion of Globally Unique EIDs Within a Hierarchy, e.g., ORG ID In Legacy Systems

Because the EID is not the primary key, the organization that owns the legacy system may choose to insert EIDs only in certain tables (e.g., the independent entities). Figure A-7 above shows the case where only the supertype entity represented as DB01_TblA, has been modified to contain a field for EIDs. In such a scenario, only DB01_TblA may be directly queried using the EID value. All the subtypes may remain hidden to an external system that is allowed to query this database if the owner so chooses. Visibility of the subtype entities may be provided either by assigning EIDs also to the subtypes, or by creating an API that works in two steps. First, the EID is used to retrieve the local key (in this example the value of DB01_TblA_Key), and then the actual data query is executed using the retrieved local key value to access the supertype and subtype entities. Figure A-8 depicts the modifications that ensue when adding EIDs to all tables in DB01.

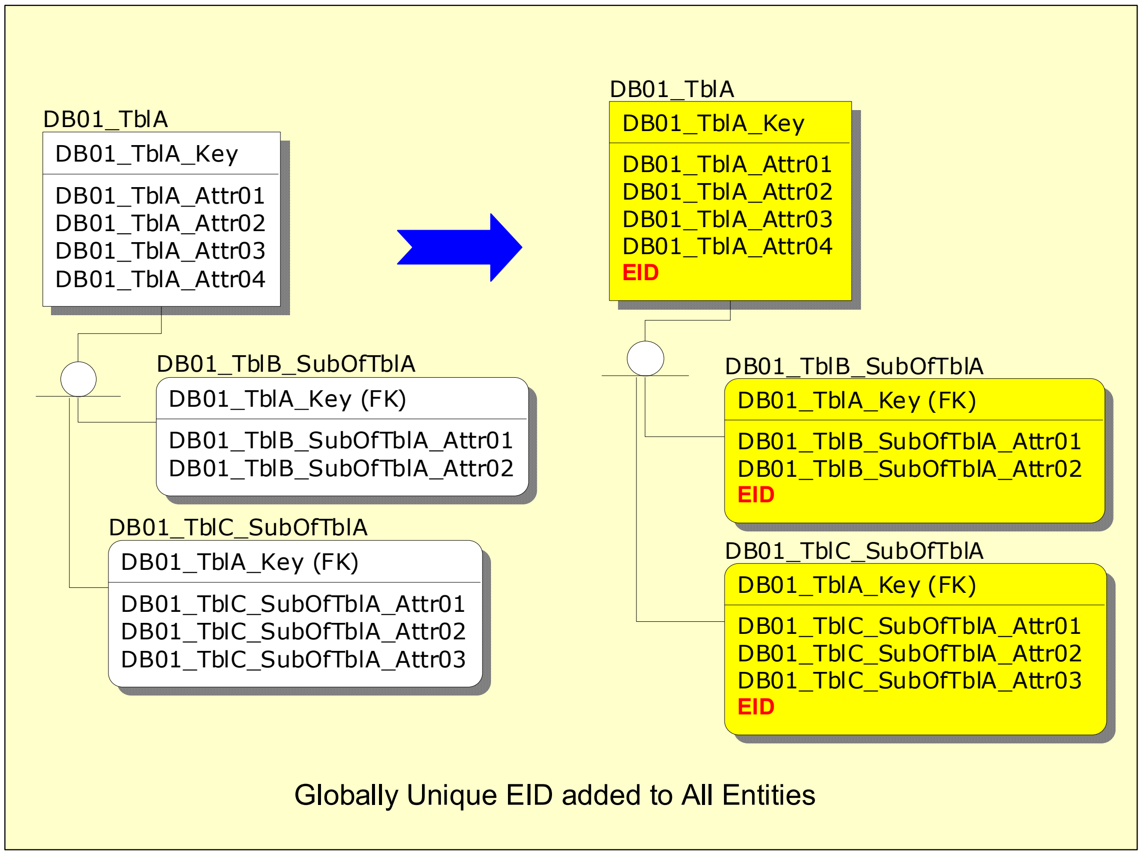


Figure A-8. Overall Insertion of Globally Unique EIDs Within a Hierarchy, e.g., ORG ID In Legacy Systems

This approach may be adequate where the database structure in DB01 has been modified so that the entity corresponding to TblB is now part of a generalization—as the local name DB01_TblB_SubOfTblA suggests. But that is not the case in other databases connecting to it. For example, in the DoD Data Architecture (DDA) the five battlefield objects (FACILITY, FEATURE, MATERIEL, ORGANIZATION and PERSON) are independent entities, but in the NATO Land C2 Information Exchange Data Model (LC2IEDM) they are subtype entities of a new entity called OBJECT-ITEM. Data collected in LC2IEDM-conformant databases may interoperate with DDA-conformant databases by using the EID to execute queries and align their data even though local key management differs among them—i.e., there is no need to query the LC2IEDM supertype entity OBJECT-ITEM prior to accessing any of the subtypes in the hierarchy, e.g., ORGANIZATION.

What has been said above for the case of supertype-subtype hierarchies applies as well when dealing with parent-child relationships. Figure A-9 shows the resulting structure when the EID is used as an alternate key only in the parent entity.

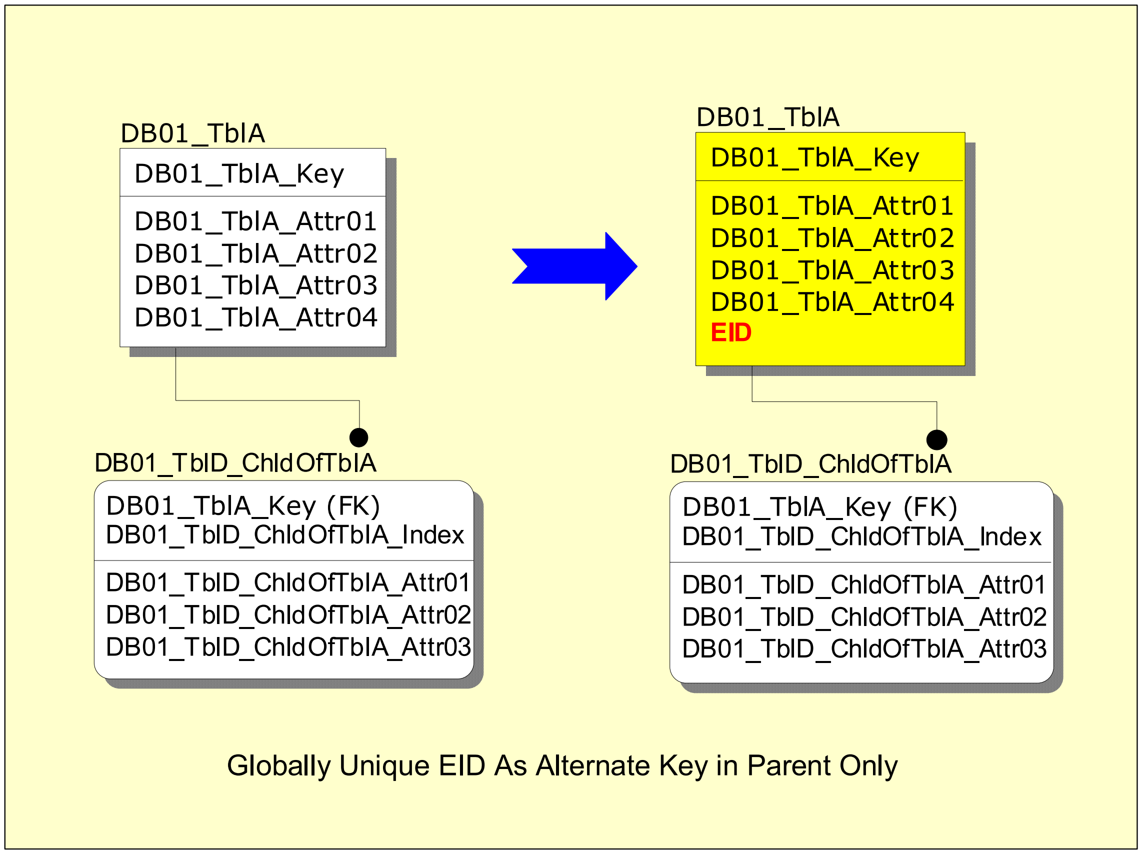

DB01_TblA
DB01_TblA_Key
DB01_TblA_Attr01
DB01_TblA_Attr02
DB01_TblA_Attr03
DB01_TblA_Attr04
DB01_TblD_ChldOfTblA
DB01_TblA_Key (FK)
DB01_TblD_ChldOfTblA_Index
DB01_TblD_ChldOfTblA_Attr01
DB01_TblD_ChldOfTblA_Attr02
DB01_TblD_ChldOfTblA_Attr03
DB01_TblA
DB01_TblA_Key
DB01_TblA_Attr01
DB01_TblA_Attr02
DB01_TblA_Attr03
DB01_TblA_Attr04
EID
DB01_TblD_ChldOfTblA
DB01_TblA_Key (FK)
DB01_TblD_ChldOfTblA_Index
DB01_TblD_ChldOfTblA_Attr01
DB01_TblD_ChldOfTblA_Attr02
DB01_TblD_ChldOfTblA_Attr03
Globally Unique EID As Alternate Key in Parent Only

Figure A-9. Partial Insertion of Globally Unique EIDs, e.g., ORG ID, Within Parent-Child Identifying Relationships In Legacy Systems

If not only the independent parent entities are meant to be accessible via EIDs, then the EID attribute can also be added as an alternate key to the child entity. Such a modification is depicted in Figure A-10. Again, consider what happens when the organization owning DB01 decides to make information contained in child entities available to external users. However, it does not want to spend time and effort adding new alternate keys to all child tables in DB01., The organization is able to share this information by providing an interface that first retrieves the value of the local key using the EID, and then executing the query based on the value of DB01_TblA_Key.

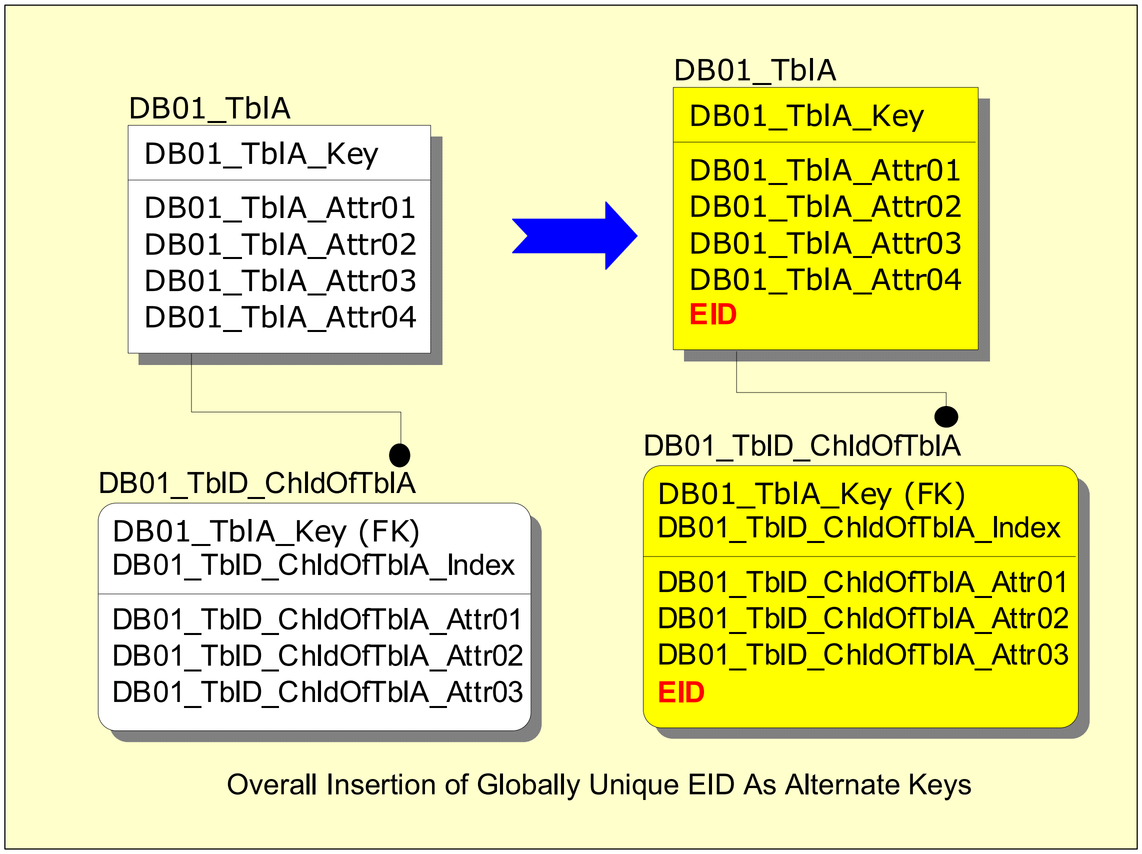

DB01_TblA
DB01_TblA_Key
DB01_TblA_Attr01
DB01_TblA_Attr02
DB01_TblA_Attr03
DB01_TblA_Attr04
DB01_TblD_ChldOfTblA
DB01_TblA_Key (FK)
DB01_TblD_ChldOfTblA_Index
DB01_TblD_ChldOfTblA_Attr01
DB01_TblD_ChldOfTblA_Attr02
DB01_TblD_ChldOfTblA_Attr03
DB01_TblA
DB01_TblA_Key
DB01_TblA_Attr01
DB01_TblA_Attr02
DB01_TblA_Attr03
DB01_TblA_Attr04
EID
DB01_TblD_ChldOfTblA
DB01_TblA_Key (FK)
DB01_TblD_ChldOfTblA_Index
DB01_TblD_ChldOfTblA_Attr01
DB01_TblD_ChldOfTblA_Attr02
DB01_TblD_ChldOfTblA_Attr03
EID
Overall Insertion of Globally Unique EID As Alternate Keys

Figure A-10. Overall Insertion of Globally Unique EIDs, e.g., ORG ID, Within Parent-Child Identifying Relationships In Legacy Systems

When the relationship among the entities is non-identifying, then insertion of EIDs as alternate keys has very little effect on the key management scheme of the legacy system, since the records in the child entity do not depend on the key of the parent entity for their identification. Whether the EID is inserted in some but not all independent entities, connected to each other via non-identifying relationships, is open to the particular data exchange needs specified for the given database, e.g., DB01. This case is schematically shown in Figure A-11, below.

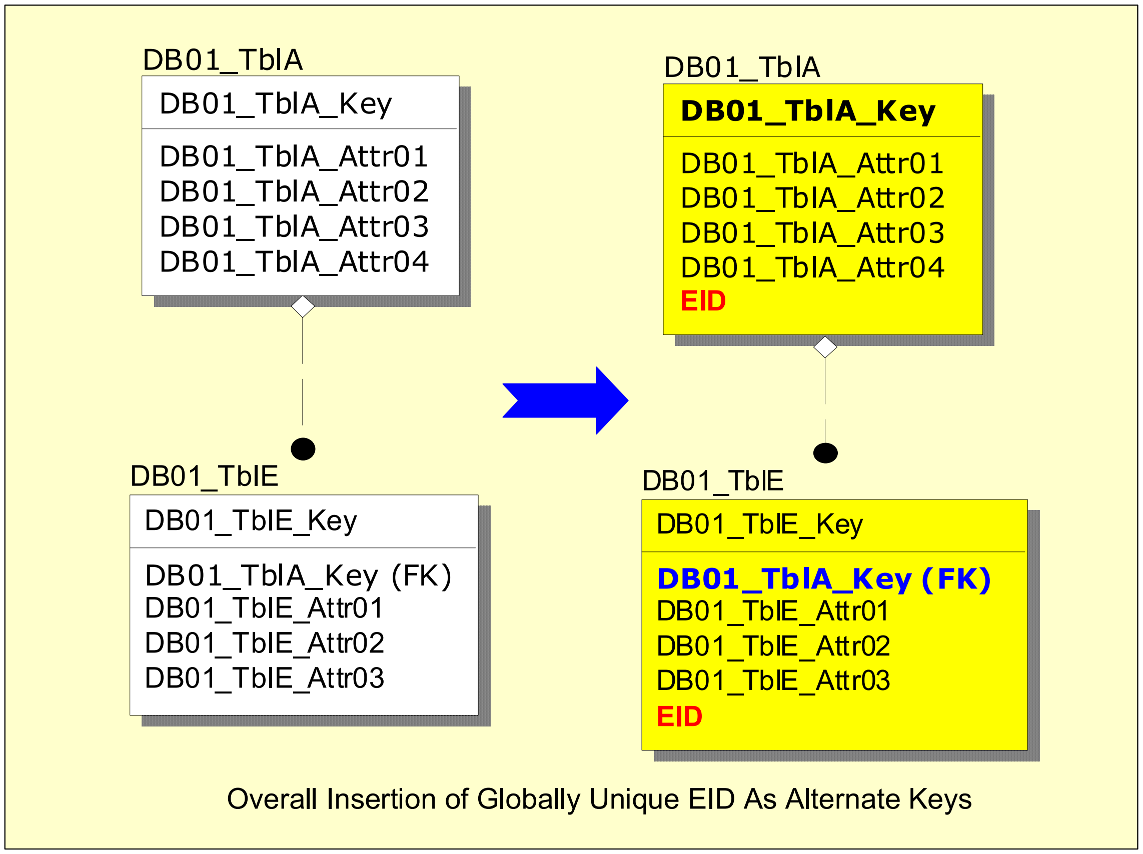


Figure A-11. Overall Insertion of Globally Unique EIDs, e.g., ORG ID, Within Parent-Child Non-Identifying Relationships In Legacy Systems

When dealing with double associative entities (see Figure A-12 below), it may be sufficient to insert the EID only in the parent entity. Normally, queries involving a parent and its double associative entity do not assume that one knows both migrated keys. In other words, the query is executed to find out which instances of Entity A are associated with another instance of the same table whose key one already knows.

In contrast, adding an EID to the double associative table permits the owner of a database, such as our notional DB01, to exercise tighter control on which associations are made available to external users and which are kept internal to the organization. The API for external users may be constructed so that it works only with the double associative entity EID value provided by the external user to the API. External users are not permitted to query the local key to traverse all possible associations that a given instance of Entity A may have beyond those that are tagged with an EID. Note that Figure A-12 depicts the modification needed to add an EID only to the parent of the double associative table.

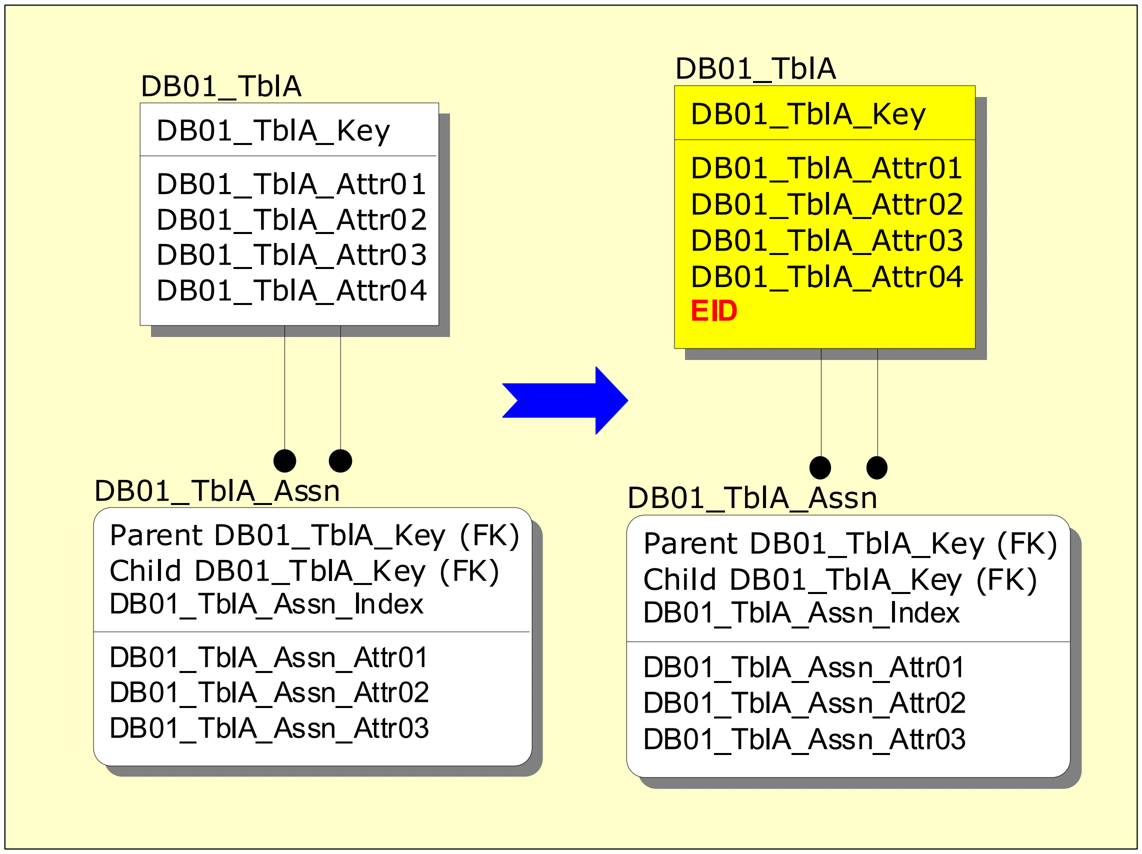


Figure 12. Partial Insertion of Globally Unique EIDs, e.g., ORG ID, Within Double-Associative Parent-Child Relationships In Legacy Systems

The previous statements apply equally to the case of regular associative entities—also known as many-to-many breakers. The only change in Figure A-12 would be that instead of a single parent entity, i.e., DB01_TblA, there would be two parent entities (for instance DB01_TblA and DB01_TblG). To execute a query, one would have to know the local key value in each table, since the associative entity can only be traversed using the value of either DB01_TblA_Key or DB01_TblG_Key. In this case, inserting an EID in the associative entity may provide some benefit, since the owner of DB01 can publish those EIDs marked for exchange while leaving others only for internal use.

2. USE OF EIDs AS PRIMARY KEYS IN EXISTING SYSTEMS

A more radical change to the key management of a legacy system wanting to use EIDs is depicted in Figure A-13. Under this approach the current primary keys are demoted to alternate key status, and all entities now use EIDs as their primary keys. The advantage of this approach is that, for example, in the case of generalizations all the subtypes automatically inherit the key of the supertype. This would mean that under this approach all SQL queries in our notional database DB01, written against the former primary key DB01_TblA_Key, can still run. Also, external systems that know the respective EID values implemented for each table can now navigate through the entire hierarchy without needing to know the values of the local alternate key. Of course, porting the data from the previous version of DB01 to the modified version may require careful attention to how the migrated attributes are reloaded so that referential integrity can be maintained (See Section E below on how this can be done).

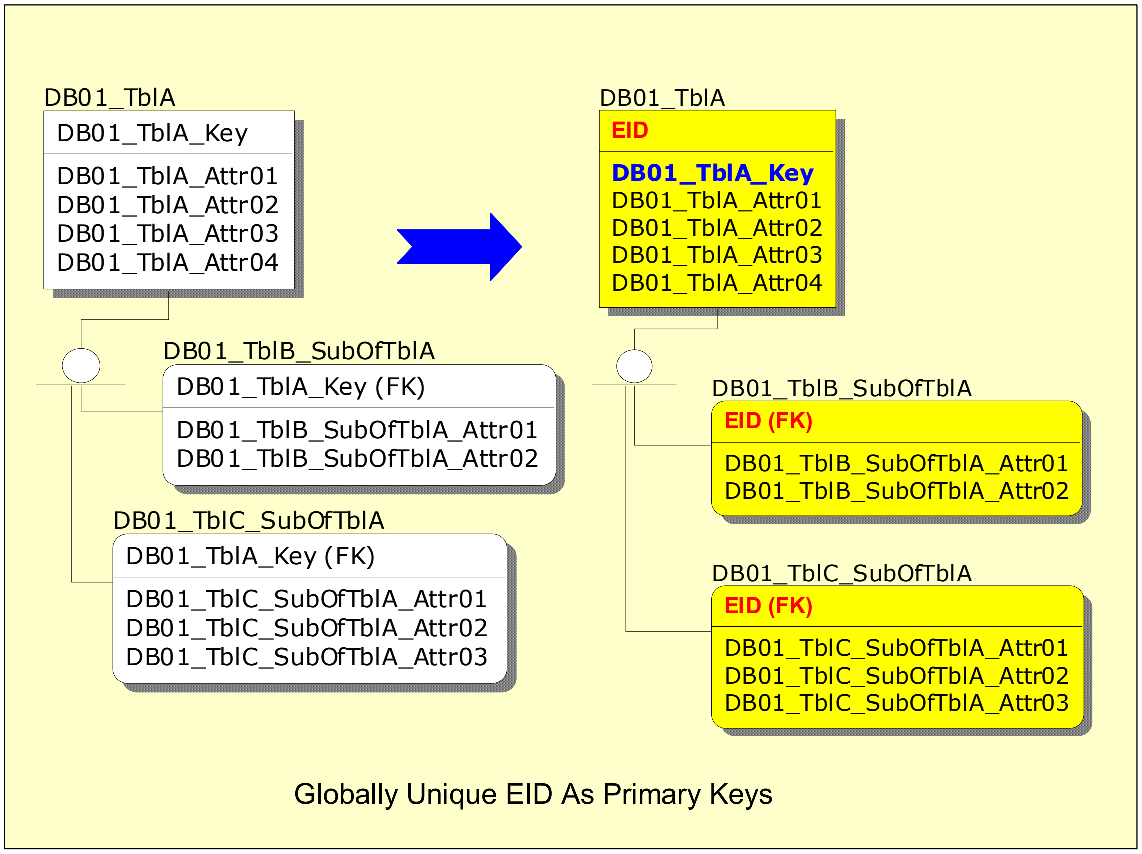


Figure A-13. Globally Unique EIDs, e.g., ORG ID, as Primary Keys for Generalizations In Legacy Systems

In the case of parent-child identifying relationships, the use of EIDs as the primary keys would also allow querying the data in DB01 in a more straightforward manner. There would be no need to access the local key in order to traverse the records of the child entity if no EID has been added there. For backwards compatibility, it may be necessary to retain the local key as an alternate key, both in the parent as well as the child entity so that existing code written against those values can still be used. Figure A-14 depicts the modification to the tables in DB01 that would take place when inserting the EID into the parent-child relationship shown therein.

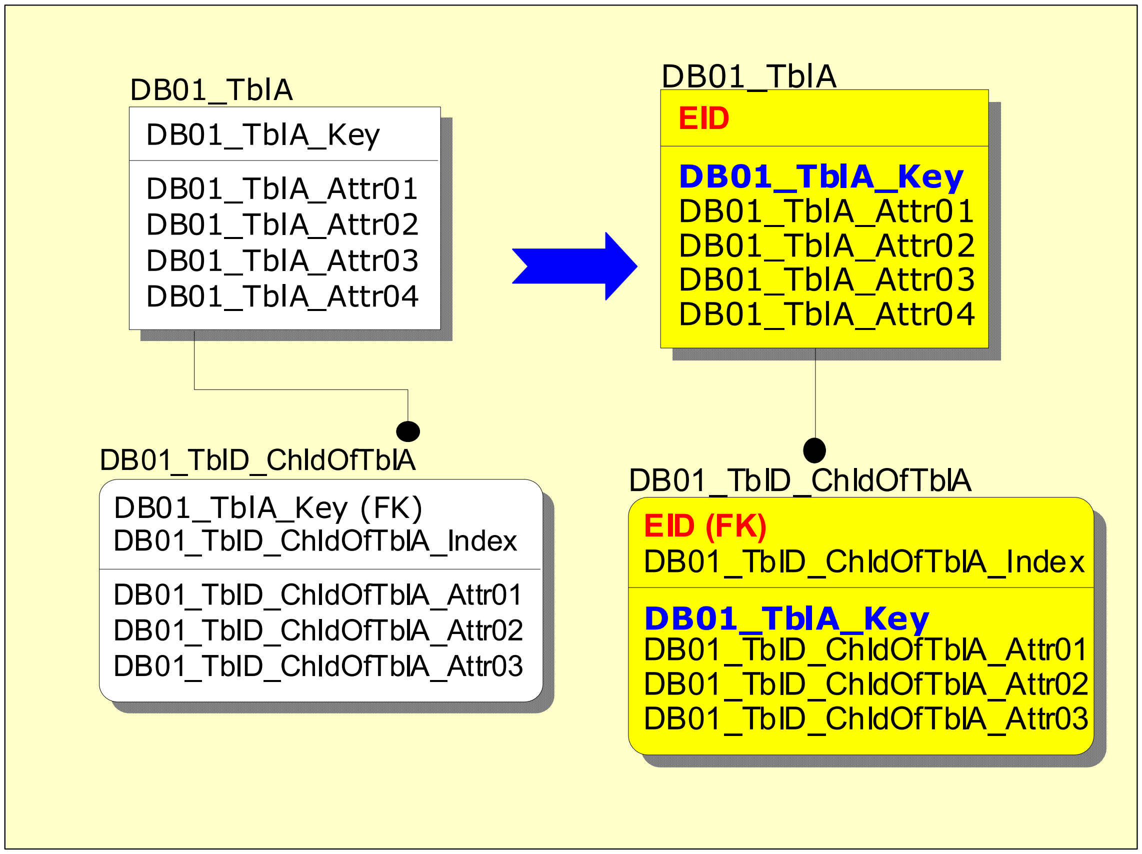


Figure A-14. Globally Unique EIDs, e.g., ORG ID, as Primary Keys for Identifying Parent-Child Relationships In Legacy Systems

Introducing EIDs as the primary keys for associative entities follows a similar approach, namely, the retention of the original local key as an alternate key and the addition of an EID to replace it. For backwards-compatibility with code written against those keys, the associative entity may retain those attributes as well. Figure A-15 shows the resulting structure for this case.

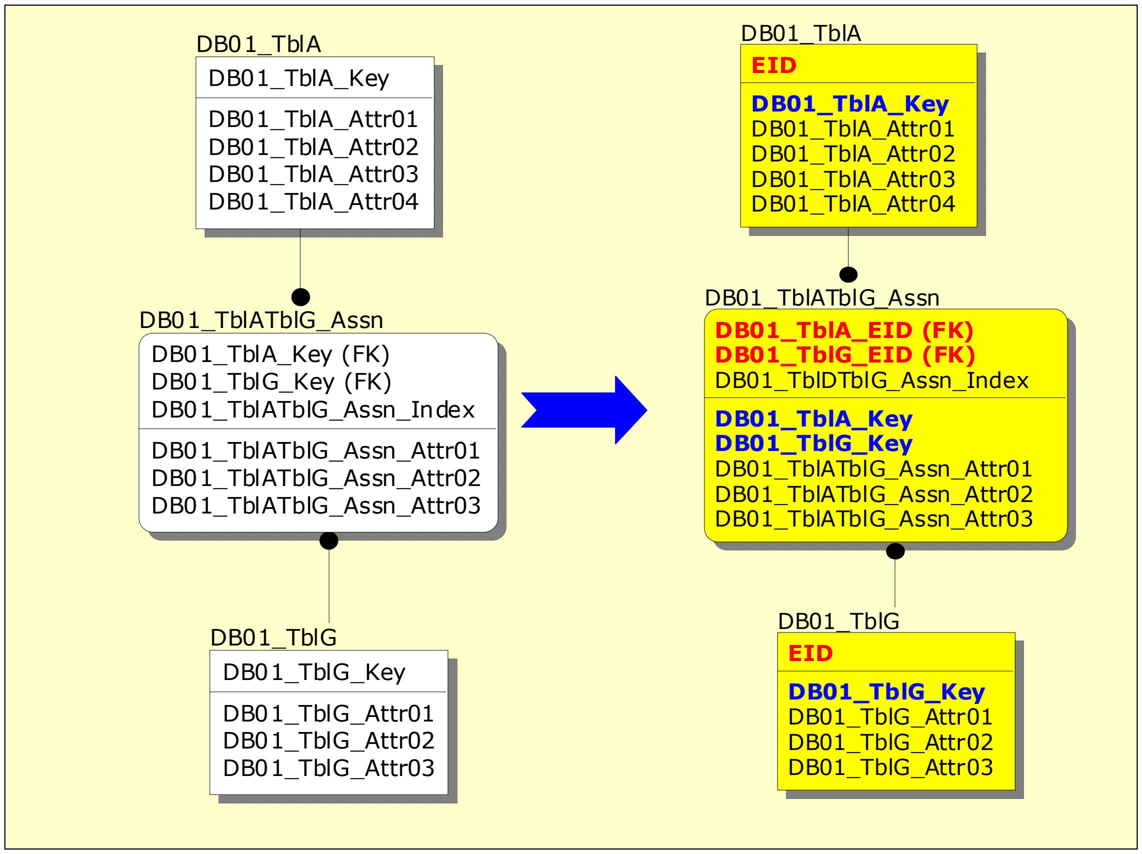


Figure A-15. Globally Unique EIDs, e.g., ORG ID, as Primary Keys for Associative Relationships In Legacy Systems

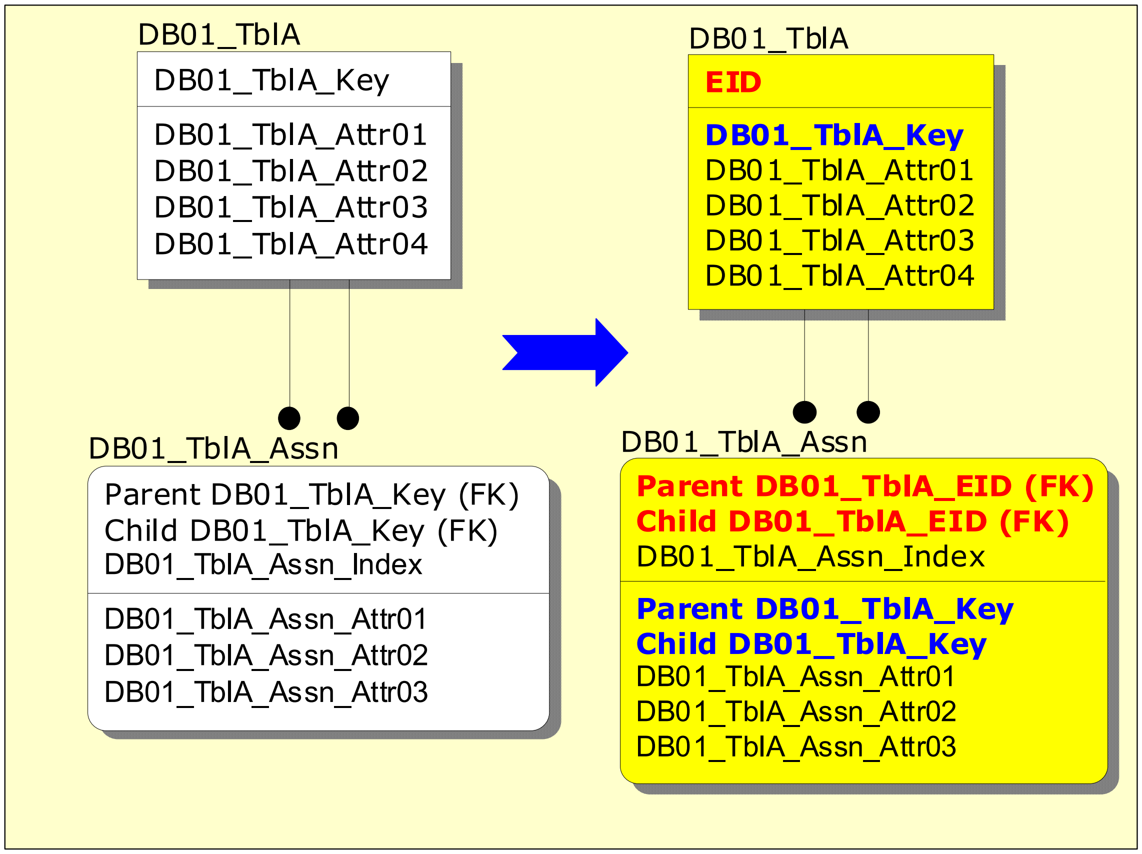


Figure A-16. Globally Unique EIDs, e.g., ORG ID, as Primary Keys for Double Associative Relationships In Legacy Systems

The transformation of the database structure for double associative entities shown in Figure A-16 is almost identical to the one shown in Figure A-15 for regular associative entities. Note that the alternate keys are kept with their original role names in order to allow reuse of code written against the original structure with minimal impact. Although nothing prevents the insertion of an EID in the double associative entity DB01_TblA_Assn, the extra step should be considered only where direct access to the records in that table via the EID is planned. As stated above, this modification permits the DB01 owner to exercise further control over the records maintained in those tables. However, similar controls may be implementable via filters in the API so that only some but not all associations are visible to an external user who knows what the EID values are.

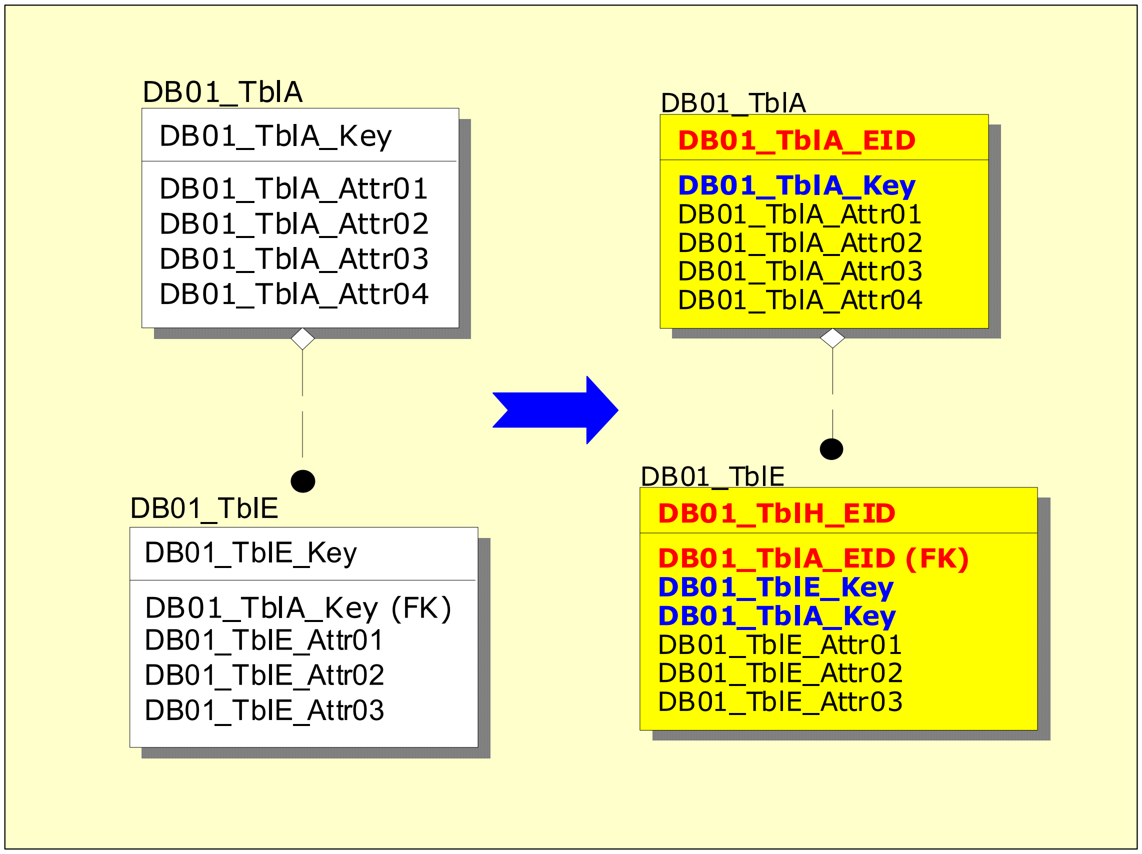


Figure A-17. Globally Unique EIDs, e.g., ORG ID, as Primary Keys for Non-Identifying Parent-Child Relationships In Legacy Systems

Under the approach we have been discussing so far, adopting EIDs in the case of non-identifying parent-child relationships does not represent any major change. Tthe child table will contain a migrated key from the parent entity just as before. The only difference is that whereas before this key was just a locally unique key, now it is an EID. As indicated for the previous examples, the local keys may be maintained as alternate keys both in the parent as well as the child entity to minimize the impact on pre-exiting code written against the locally unique keys.

It is debatable whether one should maintain the old keys indefinitely, or whether at some point it would be more efficient to make the full transformation based on EIDs only. Consider, as examples, when the volume of code written against the new primary keys, i.e., EIDs, is substantially larger than the previous code, and the cost of maintaining both the old and the new code is larger than re-writing the old code in terms of the new primary keys. This approach is illustrated in Figure A-17. Appropriate DoD-wide migration plans requiring that by some given date all data retrieval be based on EIDs may be necessary, both to give the owners of legacy systems a concrete target, as well as to allocate resources for achieving that goal without major dislocations or loss of service.

C. TECHNICAL ALTERNATIVES FOR THE USE OF EIDs IN NEW SYSTEMS

The following are the possible ways for implementing EIDs within new (i.e., planned but not yet operational) information systems:

- Retain the traditional structure of relational databases, i.e., identifying parent-child and supertype-subtype relationships. Use the EIDs as primary keys but only for the parent entities, i.e., provide only sequential indexes for the child entities.
- Remove all identifying parent-child and supertype-subtype relationships. Assign EIDs to all entities.But retain non-identifying relationships among the entities.
- Remove all relationships among the entities, Assign EIDs to all the entities. Create all pertinent relationships by establishing record-level associations based on the EID values loaded in each table.

1. USE OF EIDs AS PRIMARY KEYS IN NEW SYSTEMS WITH STANDARD RELATIONSHIPS

What has been said in Section B.2 above (Figures A-13 through A-17) applies to new systems with the single difference that since these are systems designed from the beginning to use EIDs, there are, therefore, no legacy local keys to maintain and track. In other words, if DB01 had been designed as a new system with EIDs, there would be no alternate keys of the type DB01_TblA_Key, DB01_TblE_Key, etc., in any of the tables but only EIDs. The typical relationships (identifying and non-identifying parent-child relationships, associative relationships and generalizations) are all the same as before. Moreover, the RDBMS engine is used to perform all database operations based on the EIDs from the beginning—e.g., cascade deletes and updates.

This type of result is likely to be the one that will occur in Army systems that are now being designed in conformance with the Core Architecture Data Model (CADM). Currently, CADM-conformant systems in the Army are centrally managing the keys for each of the tables in the physical schema. Due to software restrictions, the identifiers are not implemented as 64 bit integers. In the FY03-FY04 period, it is expected that all RDBMSs will have this capability. At this point, 64 bit integers can be loaded, and it is expected that they will be EIDs of the type discussed in Section A under Figure A-5.

2. USE OF EIDs AS PRIMARY KEYS IN NEW SYSTEMS WITH ONLY NON-IDENTIFYING RELATIONSHIPS

The global uniqueness of the EIDs, however, may allow database designers to recast their data models in a form that greatly simplifies the primary key specification of all entities. Since every record in every table is uniquely identifiable within the entire database rather than just locally, all entities can be designed without any need for composite primary keys. In addition, since supertype-subtype relationships are equivalent to Z-relationships, they too can be reformulated in this form. Figure A-18 below shows the equivalent formulation of a supertype-subtype hierarchy in the form of non-identifying Z-relationships. In order to ensure that no record in the child entity can be created that does not have a corresponding record in the parent entity (the supertype), the relationship is defined as no-nulls-allowed.

On the one hand, although this approach is quite elegant, arguably there are two EIDs which could uniquely identify each record in the subtype entities. For this reason, one may consider relaxing the rule that only non-identifying relationships should be used under this approach and allow their use exclusively for generalizations.

Figure A-19 below depicts the resulting structure when using identifying Z-relationships to express supertype-subtype hierarchies. Since within IDEF1X a subtype hierarchy is understood as a decomposition into mutually exclusive classes, code within the RDBMS must ensure that the same instance of the supertype entity not be a Z-child in more than one of the subtype entities.

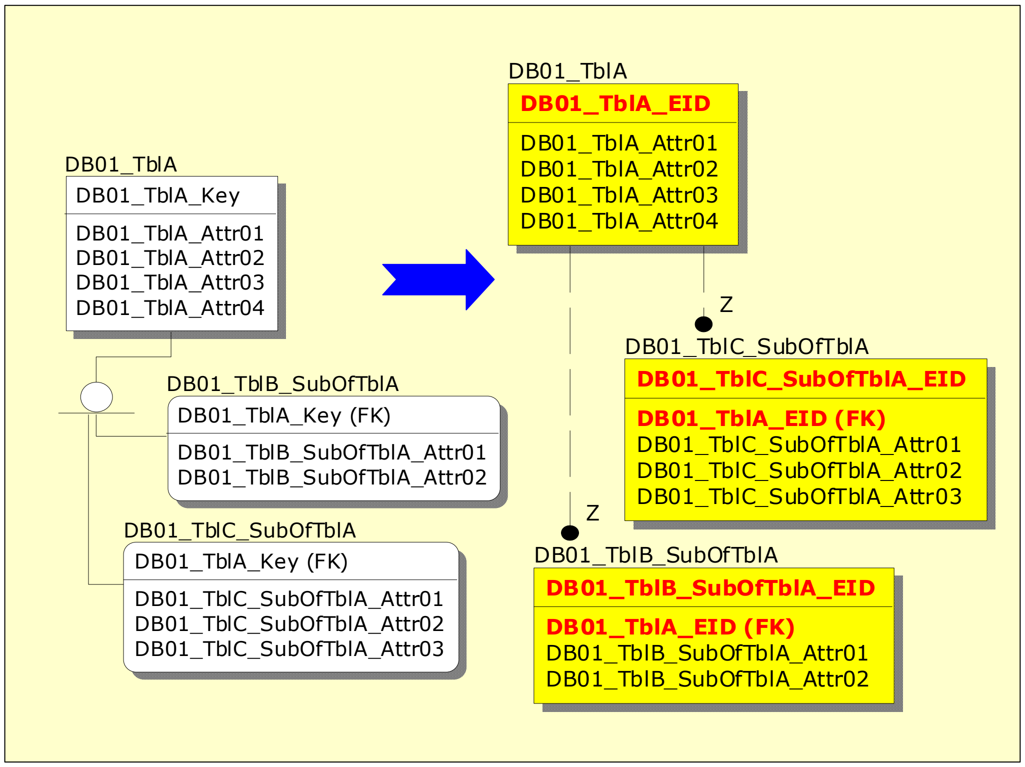


Figure A-18. Globally Unique EIDs, e.g., ORG ID, as Primary Keys with Generalizations Recast as Non-Identifying Z-Relationships

On the other hand, relaxing this rule may enable the owner of the database to create a business rule that permits the users to combine attributes from different classes without a need for an explicit new subtype—e.g., an amphibious vehicle may be specified as the combination of a land vehicle and a vehicle thus capable of moving on land and navigating in water. The application may provide the option of choosing any and all pertinent attributes from the subtype entities as required.[70]

[70] This is something quite familiar to application developers working with Object Oriented (OO) data models, and shows yet another potential benefit for the use of EIDs, namely, the ability to bridge the gap between OO and traditional Entity-Relationship Data models (ERD).

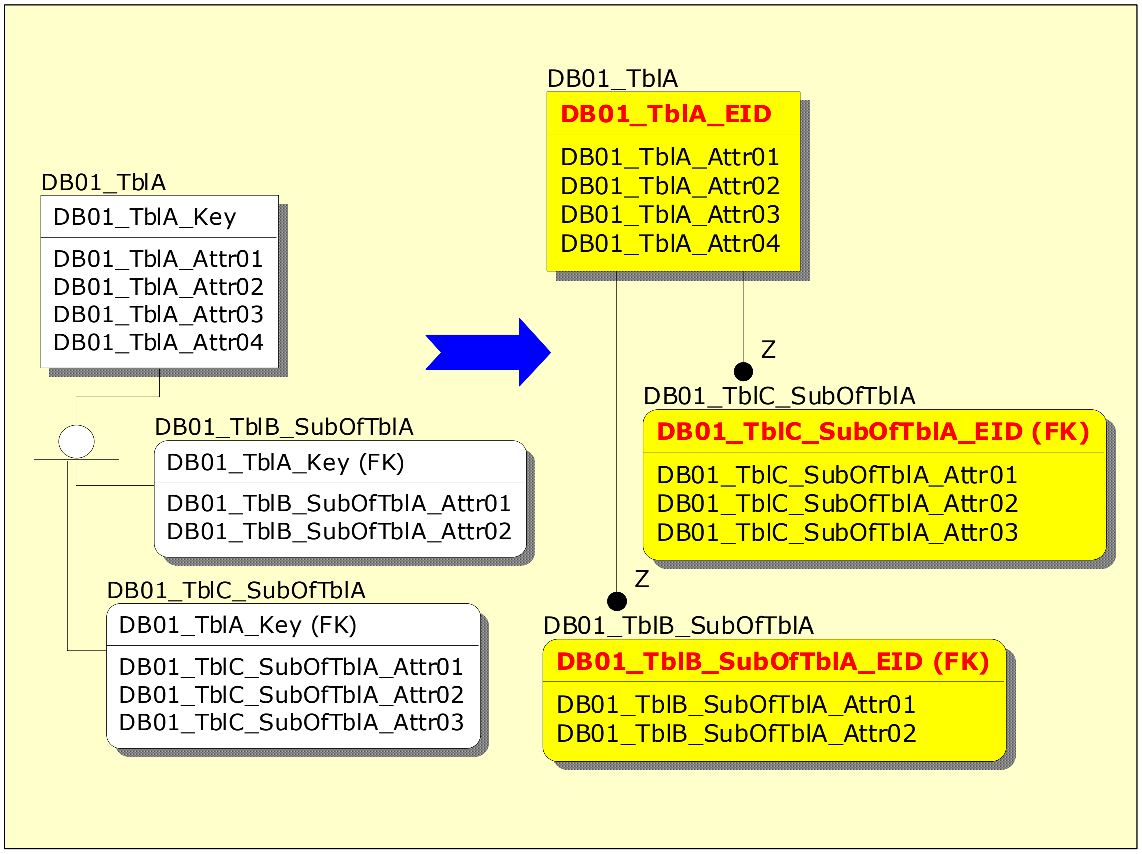

DB01_TblA
DB01_TblA_Key
DB01_TblA_Attr01
DB01_TblA_Attr02
DB01_TblA_Attr03
DB01_TblA_Attr04
DB01_TblB_SubOfTblA
DB01_TblA_Key (FK)
DB01_TblB_SubOfTblA_Attr01
DB01_TblB_SubOfTblA_Attr02
DB01_TblC_SubOfTblA
DB01_TblA_Key (FK)
DB01_TblC_SubOfTblA_Attr01
DB01_TblC_SubOfTblA_Attr02
DB01_TblC_SubOfTblA_Attr03
DB01_TblA
DB01_TblA_EID
DB01_TblA_Attr01
DB01_TblA_Attr02
DB01_TblA_Attr03
DB01_TblA_Attr04
Z
DB01_TblC_SubOfTblA
DB01_TblC_SubOfTblA_EID (FK)
DB01_TblC_SubOfTblA_Attr01
DB01_TblC_SubOfTblA_Attr02
DB01_TblC_SubOfTblA_Attr03
Z
DB01_TblB_SubOfTblA
DB01_TblB_SubOfTblA_EID (FK)
DB01_TblB_SubOfTblA_Attr01
DB01_TblB_SubOfTblA_Attr02

Figure A-19. Globally Unique EIDs, e.g., ORG ID, as Primary Keys with Generalizations Recast as Identifying Z-Relationships

There is no substantial gain in recasting identifying relationships as non-identifying relationships with EIDs as the primary key except that, as mentioned above, the keys consist of a single attribute—i.e., there are no composite keys under this approach. Figure A-20 above depicts the resulting structure under this approach. The possible benefit arising from this approach may be in performance gains. The table records need only one attribute rather than the migrated key plus an index under the traditional approach. It is also clear, once this approach is adopted, both traditional identifying and non-identifying relationships can be mapped to the same EID-based data structure.

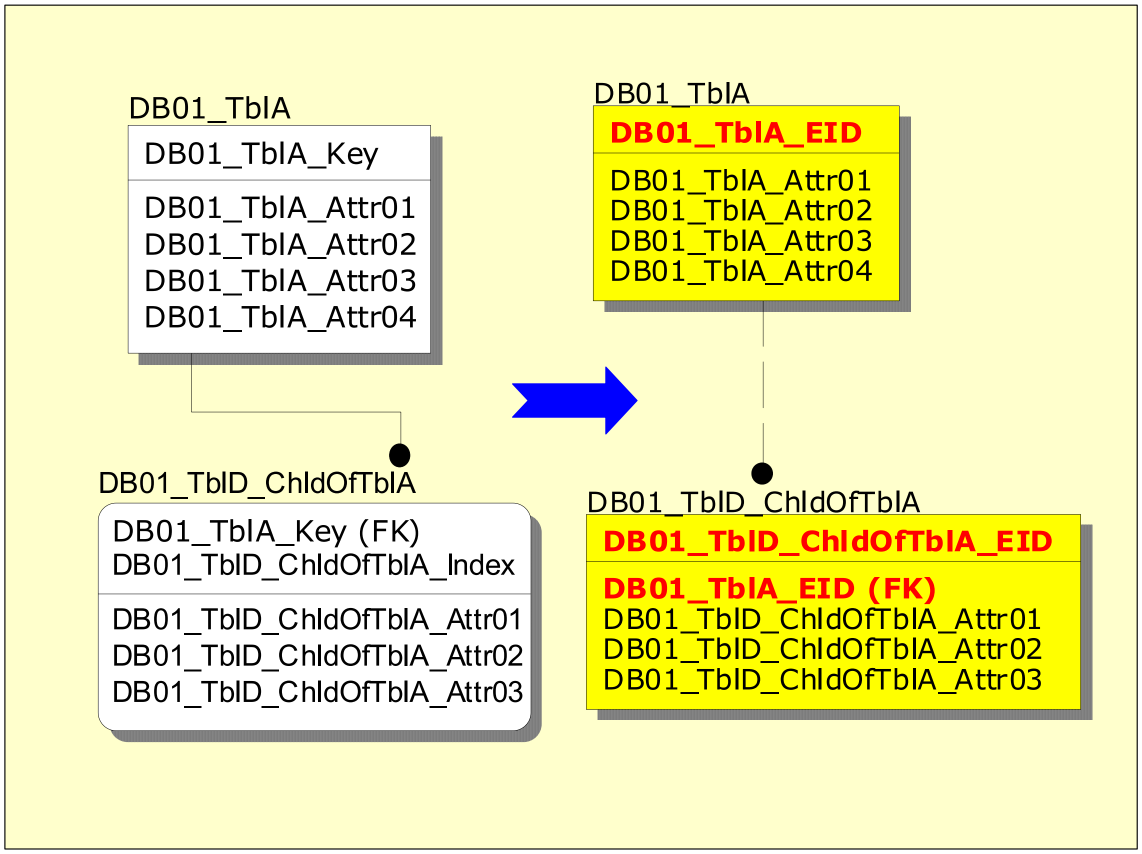


Figure A-20. Globally Unique EIDs, e.g., ORG ID, as Primary Keys with Identifying Relationships Recast as Non-Identifying-Relationships

Adding another parent entity to the diagram in Figure A-20 to represent a many-to-many breaker entity results in a structure with two EIDs below the primary key area, namely DB01_TblA_EID and DB01_TblG_EID (See Figure A-21 below). The potential benefit of such a use of EIDs is the same as the one for parent-child relationships mentioned above, i.e., performance gains. The table records need only one attribute rather than the migrated key plus an index under the traditional approach.

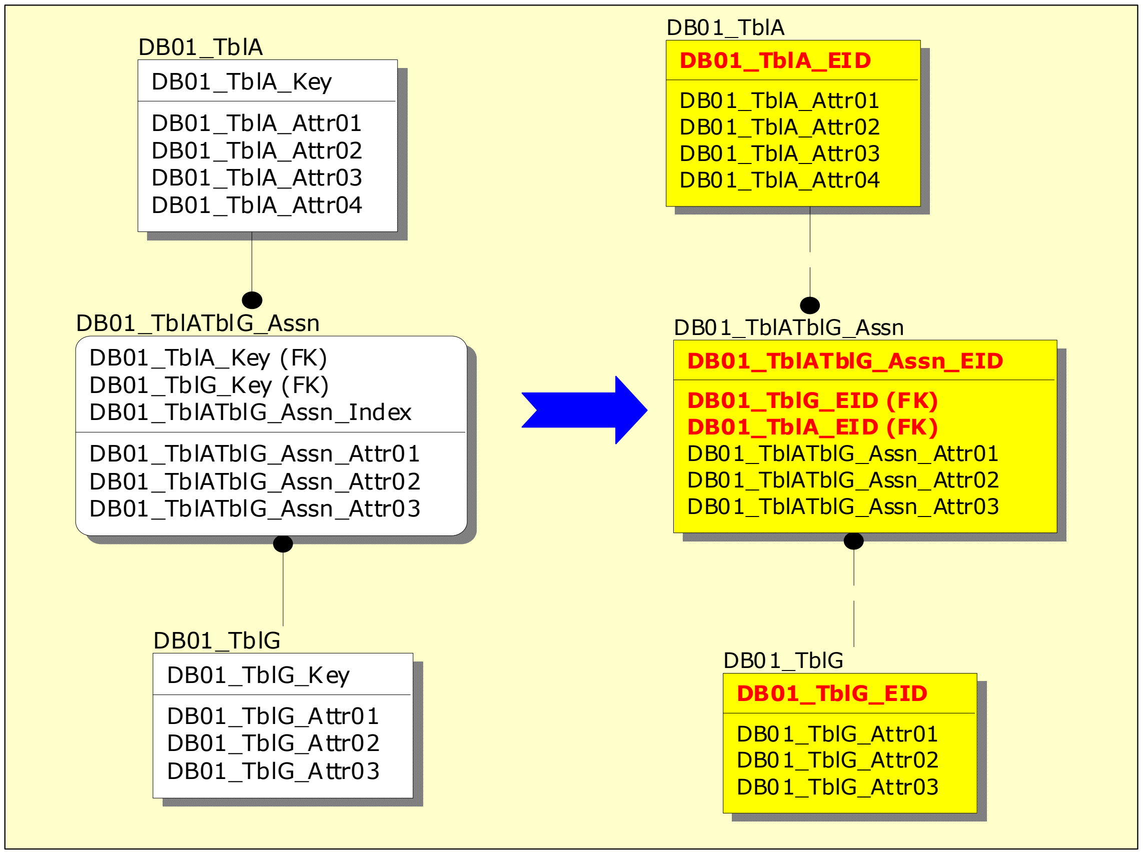


Figure A-21. Globally Unique EIDs, e.g., ORG ID, as Primary Keys with Identifying Relationships in an Associative Entity Recast as Non-Identifying-Relationships

There is no substantial difference between this use of EIDs in many-to-many breaker entities such as the one shown in Figure A-21 and the resulting structure from a double associative entity using non-identifying relationships. Instead of having two EIDs coming from two different entities, the double associative entity would also be independent, but would be related to a single parent entity and the two migrated keys. Below the primary key area would be the role-named keys from the parent entity. Again, the potential benefit may lie in performance gains, since the table records need only one attribute rather than the two migrated keys plus an index under the traditional approach

3. USE OF EIDs AS PRIMARY KEYS IN NEW SYSTEMS WITH ALL RELATIONSHIPS CAPTURED VIA DATA

The previous sections have shown that the insertions of globally unique EIDs, such as the ORG ID, allow the handling of data in ways that are not possible when the keys are only unique at the table level—the typical situation in most current database implementations.

Nevertheless, in all the alternatives discussed in Section 2 above, we have made use of the traditional key migration from parent entity to child entity. As a result, the RDMBS engine can perform the required table joins and execute standard SQL data manipulation operations using such pointers embedded in the dependent tables. Thus, in all the cases surveyed above, child entities, even when modeled as independent entities with the parent key below the primary key area, still have another EID. This EID theoretically could identify all the records in the table (i.e., by reverting to the use of a sequential index coupled to the EID to permit the many-children.)

A completely different approach looks at data entities within any given database using EIDs—such as the notional DB01 used in the examples above—more like objects within an object-oriented approach (the term is used loosely here). This approach results in the complete elimination of all relationships among the logical entities and the use of both EIDs as the single primary key for their records. Moreover, a physical database table captures the relationships at the record level between any record from any table to any other record in the same table or in a different one.

Figure A-22 shows conceptually what the table structure of the notional database DB01 would look like under this approach. Note that here the entity relationships are meant to be captured via data in DB01_MasterLookup (the blue table in Figure A-22).

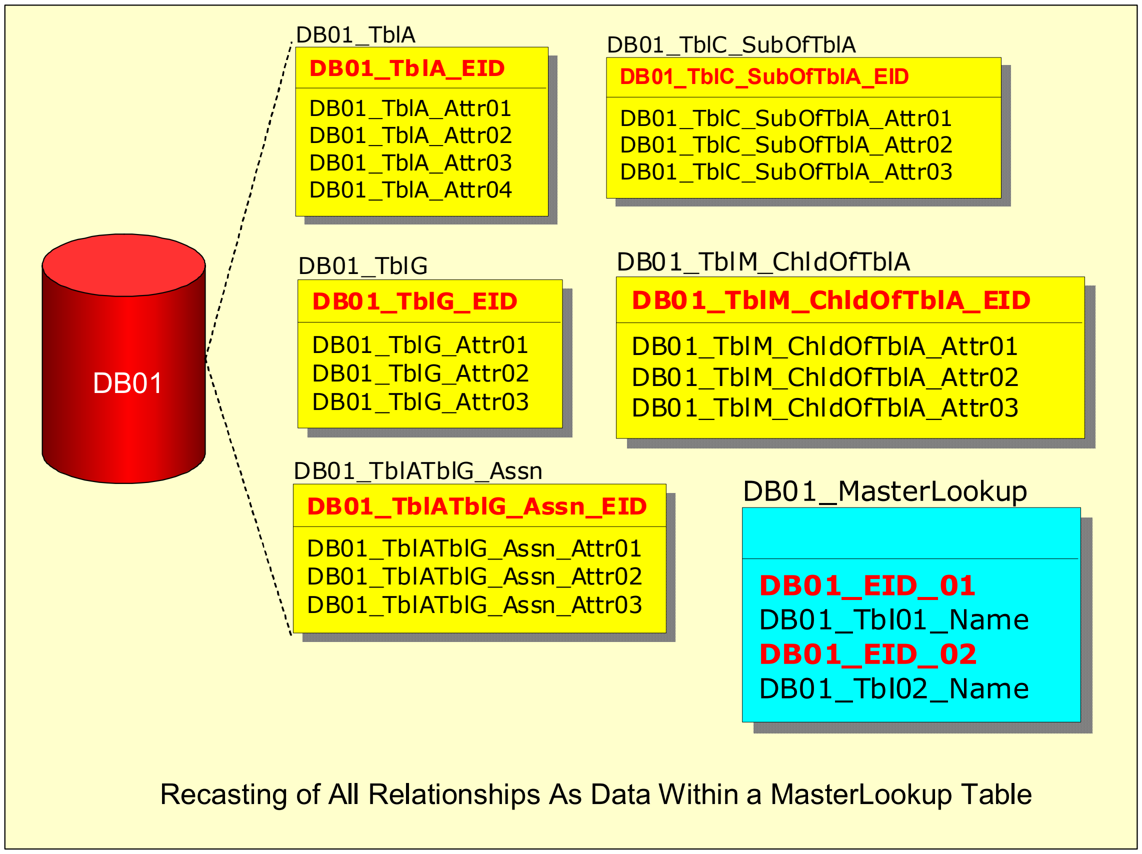


Figure A-22. Globally Unique EIDs, e.g., ORG ID, as Primary Keys and Recasting of all Relationships as Record-Level Entries within a Master Lookup Table

The question that needs to be answered now is whether all the typical IDEF1X structures can be adequately implemented using this approach. In other words, one needs to show that the following IDEF1X constructs can be faithfully replicated using the new approach, namely, (1) identifying Parent-Child relationships of any depth, (2) non-identifying Parent-Child relationships, (3) generalizations of any depth, (4) associations—also called many-to-many breakers and (5) double associative entities. The following paragraphs schematically show how each one of these IDEF1X constructs can be recast in terms of globally unique EIDs, coupled with the use of a lookup table that encodes at the record-level all the relationships types listed previously.

IMPLEMENTATION OF PARENT-CHILD RELATIONSHIPS VIA EIDS AND THE MASTERLOOKUP TABLE

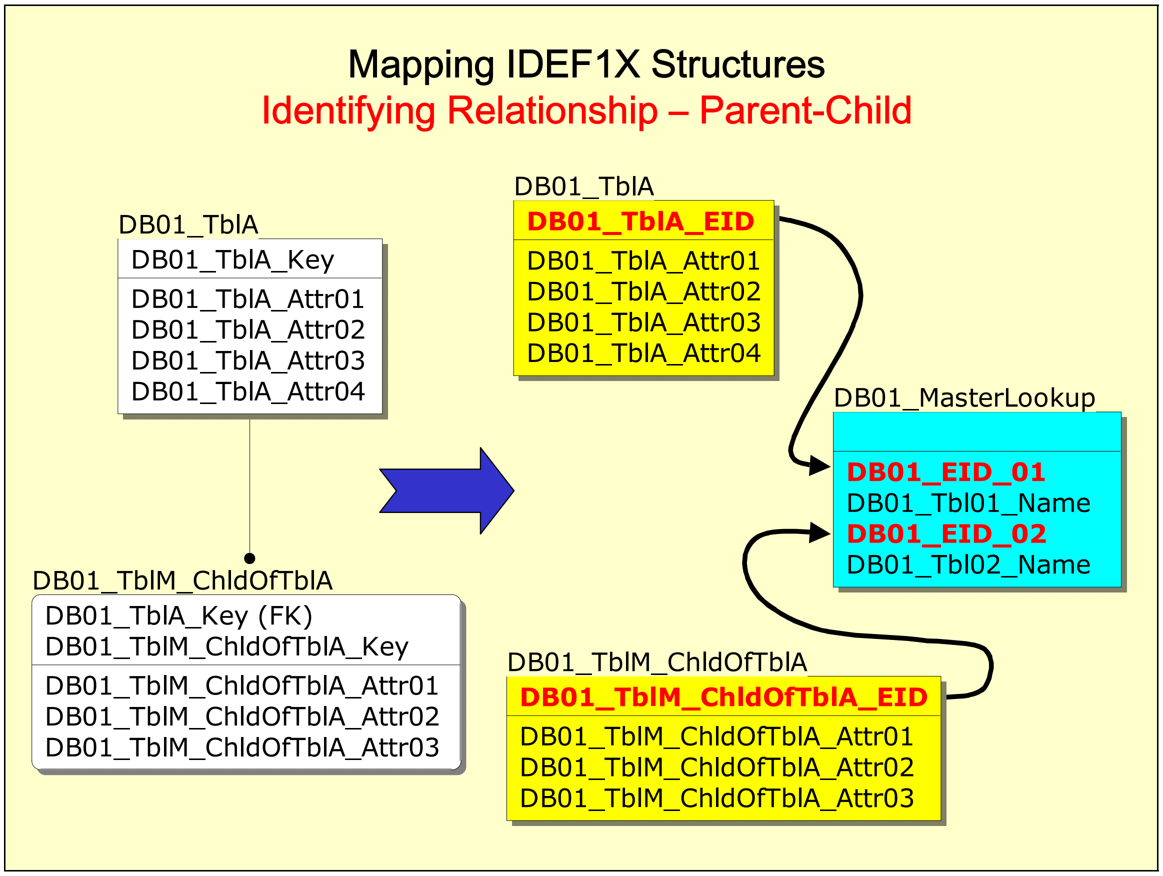


Figure A-23. Any Identifying Parent-Child Relationship Maps to a Record-Level Entry within the MasterLookup

As was highlighted above, under this new approach for using EIDs, there is no longer a migration of keys from parent entities to child entities, since all the tables are independent and there are no relationships among them. Instead, all relationships are recorded in a lookup table. One thing to keep in mind is that business rules, previously captured in entity relations, are now established by the entries in the lookup table and the code written for the user-interfaces, forms, and queries.

Figure A-23 shows the mapping of an identifying parent-child relationship from the traditional RDBMS to this new type of EID implementation for a new system. The entity-level parent-child relationship is now recorded in the lookup table, here notionally labeled DB01_MasterLookup, as pairs of entries for those records which stand in that kind of relationship. The EID value and table name of the parent entity, DB01_TblA, are recorded in DB01_EID_01 and DB01_Tbl01_Name, respectively. Similarly, the EID value and table name of the child entity, DB01_TblM_ChldOfTblA, are recorded in DB01_EID_02 and DB01_Tbl02_Name, respectively. To find the children of a particular record in DB01_TblA, the EID value of the child record is queried against DB01_MasterLookup given the parent record EID, along with the table names of both the parent and child tables. The result of the query contains the EIDs of the children entities, which are used to identify the desired records back in the DB01_TblM_ChldOfTblA table. Therefore, an identifying parent-child relationship is converted to a single pass through the lookup table that retrieves the pertinent EIDs in the child table.

Implementation of Parent-Child-GrandChild Relationships Via EIDs and the MasterLookup Table

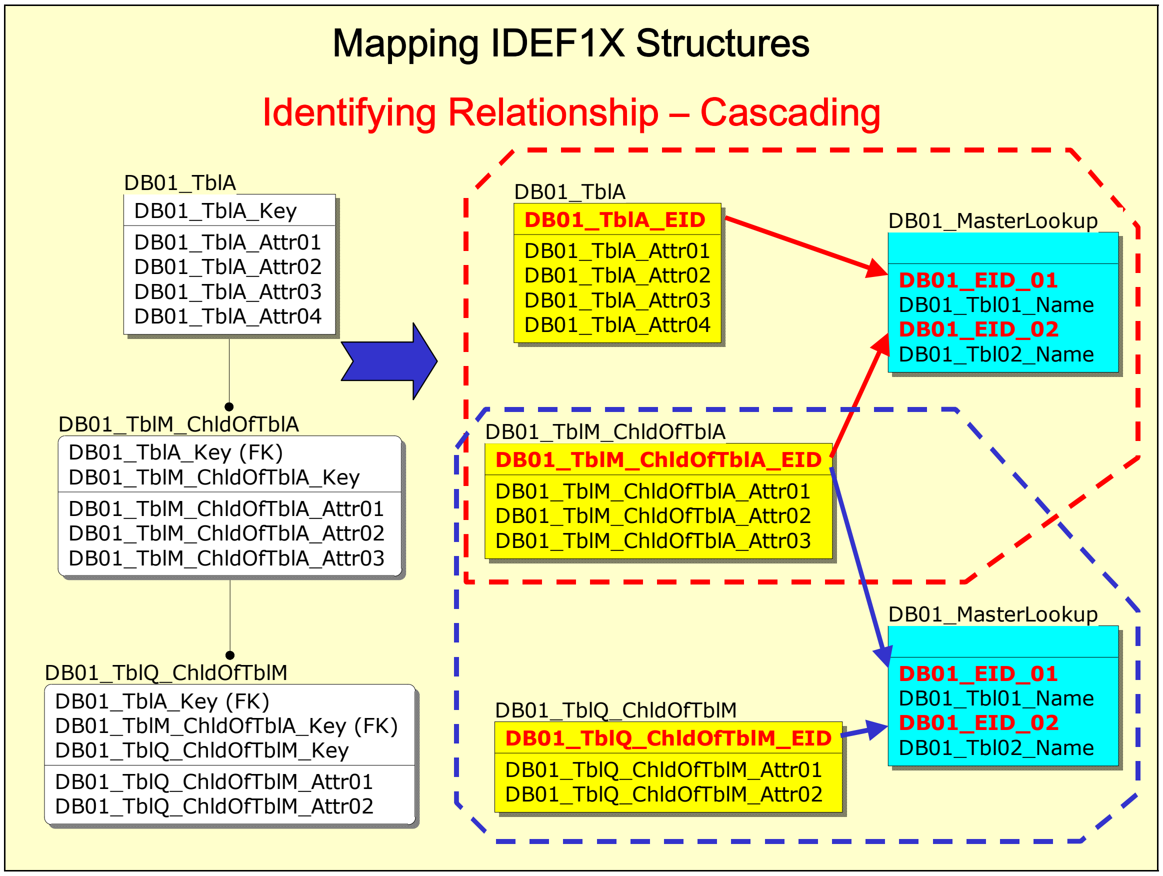


Figure A-24. Any Identifying Parent-Child-GrandChild Relationship Maps to a Record-Level Entry within the MasterLookup

Figure A-24 shows the mapping of a traditional cascading identifying relationship (left side) into an implementation without any relationships among the tables and with all records identified via globally unique EIDs (right side). This is, in essence, an extension of the method described above for the case of an identifying parent-child relationship. The cascading relationship is resolved into pairs of parent-child relationships. In this example, the first pair, containing the parent-child relationship between DB01_TblA and DB01_TblM_ChldOfTblA, is outlined in red. The second pair, containing the child-grandchild relationship between DB01_TblM_ChldOfTblA and DB01_TblQ_ChldOfTblM, is outlined in blue. The relationship between each pair is recorded in DB01_MasterLookup in exactly the same manner as it was described above for the simple identifying parent-child relationship. Because the 'relationship' information is encoded as pairs of entries in the DB01_MasterLookup table, the retrieval of data related to each other in the form of parent-child pairs can be thought of as an iterative process. Therefore, there is no limit to the levels of descendants that can be accessed via the DB01_MasterLookup table. Furthermore, any cascade operation—updates, deletes, etc.—can be done with the aid of the DB01_MasterLookup table and the appropriate SQL command in the same iterative manner for the tables and records involved.

Implementation of Non-Identifying Parent-Child Relationships Via EIDs and the MasterLookup Table

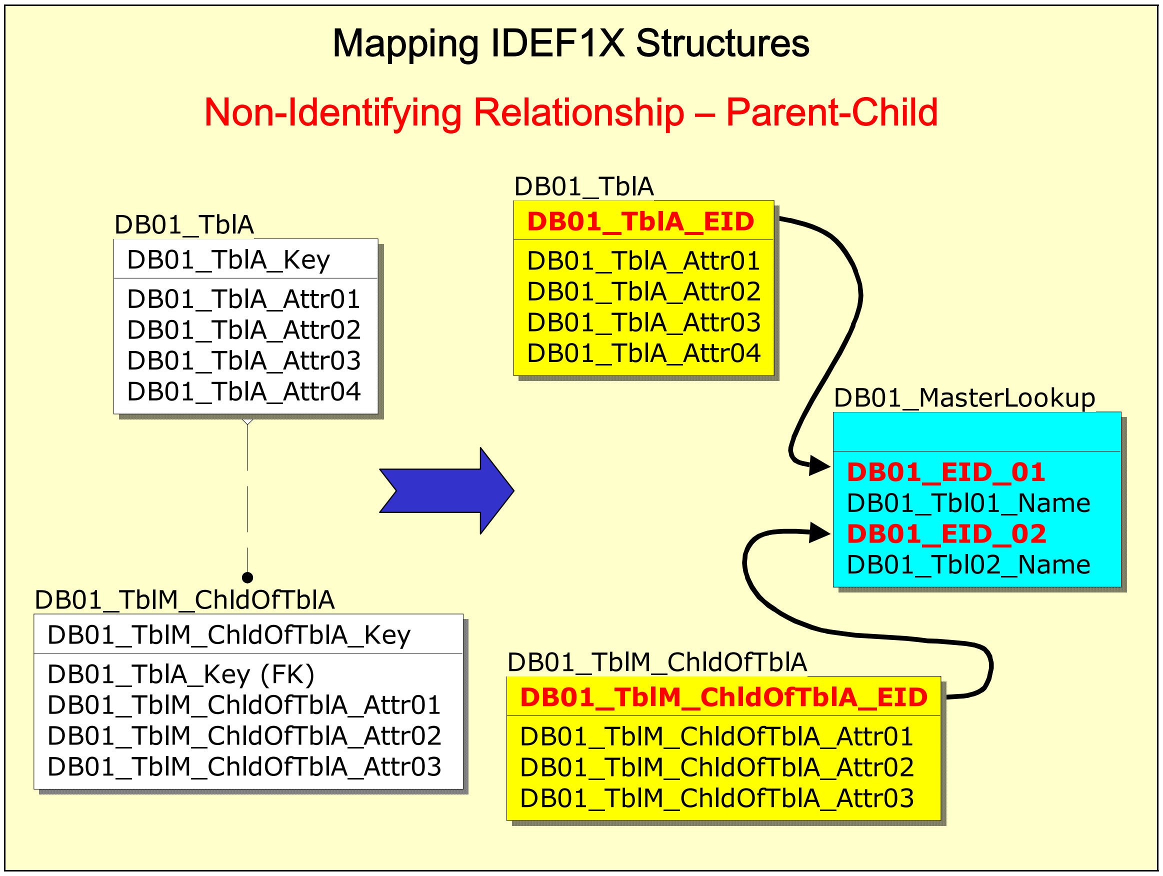


Figure A-25. Any Non-Identifying Parent-Child Relationship Maps to a Record-Level Entry within the MasterLookup

Figure A-25 shows the mapping of a non-identifying parent-child relationship. Because this new system would employ EIDs as the primary key for all entities, there is no longer a need to inherit identifiers from parent entities, that is, every record is uniquely identified by its own EID. Therefore, a non-identifying parent-child relationship can be mapped in exactly the same way as an identifying parent-child relationship where the relationship is mapped as a single line in the DB01_MasterLookup table (see the discussion above and Figure A-23).

Implementation of Generalizations of Any Depth Via EIDs and the MasterLookup Table

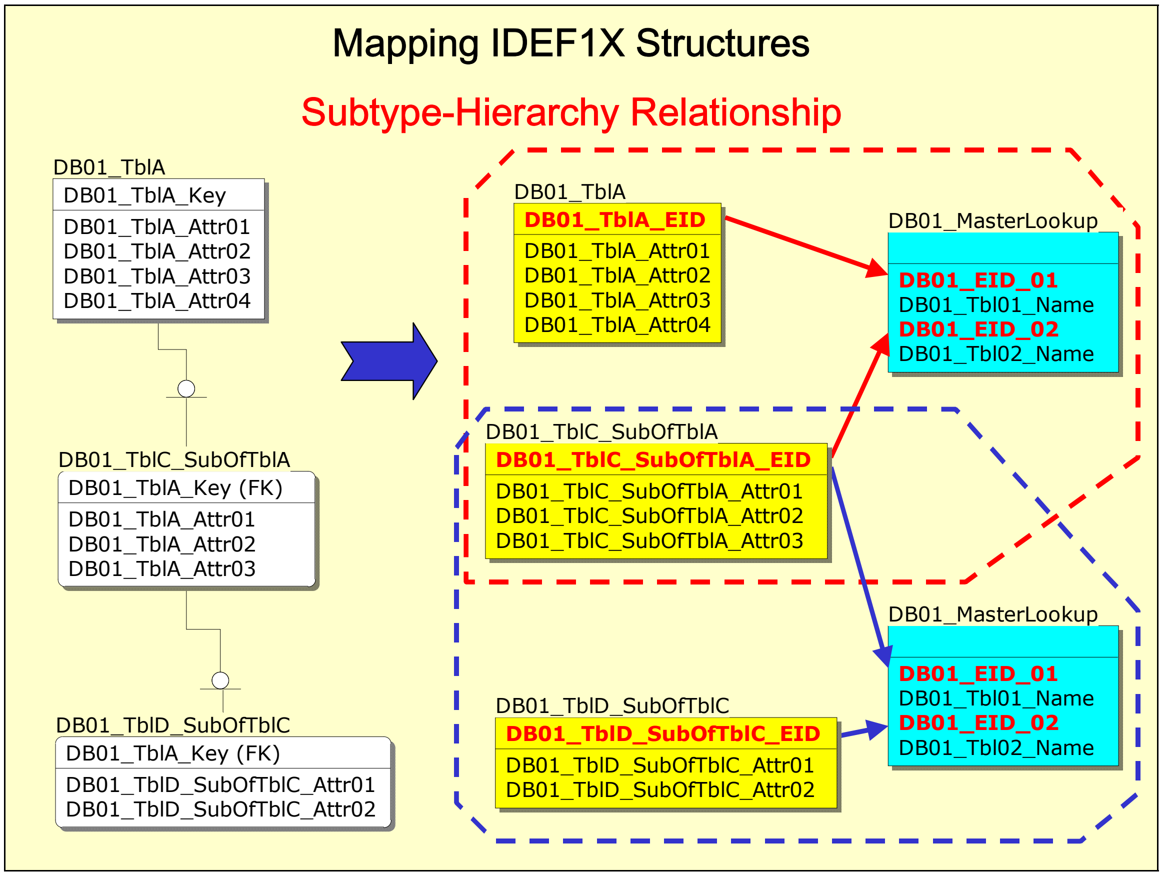


Figure A-26. Any Generalization of Any Depth Maps to a Record-Level Entry within the MasterLookup

A subtype hierarchy tree can be thought of as a cascade of Z-relationships, that is, a parent record in a subtype hierarchy has either 0 or 1 corresponding records in the child subtype entity. The mapping of this structure, shown in Figure A-26, is therefore straightforward. The subtype hierarchy is nothing more than a restricted cascading parent-child relationship identical to the one that was discussed above (See Figure A-23).

The record-keeping in DB01_MasterLookup is exactly the same as the one previously described for the cascading identifying relationship. The Z-type restriction placed on the subtype hierarchy can be controlled by the database administrator through the appropriate code in the user interfaces and forms used to perform record manipulations, e.g., inserts and deletes. Since the users can only access the data through these interfaces, the business rule can, therefore, be easily enforced and the original Z-type restriction thereby conserved.

As shown on the right hand of Figure A-26 above, a multilevel subtype hierarchy maps once again to pair-wise entries in DB01_MasterLookup where DB01_TblA supertype records are linked to subtype records in DB01_TblC_SubOfTblA. Likewise, supertype records in DB01_TblC_SubOfTblA are linked to subtype records in DB01_TblD_SubOfTblC. It should be noted, however, that the use of EIDs makes it possible, at the discretion of the database administrator, to create a link directly between any two nodes on the subtype hierarchy.In our example, one can link the root entity DB01_TblA and the leaf entity DB01_TblD_SubOfTblC with a direct entry in lookup table, DB01_MasterLookup, thus shortening the search path.[71]

Implementation of Associative Entities Via EIDs and the MasterLookup Table

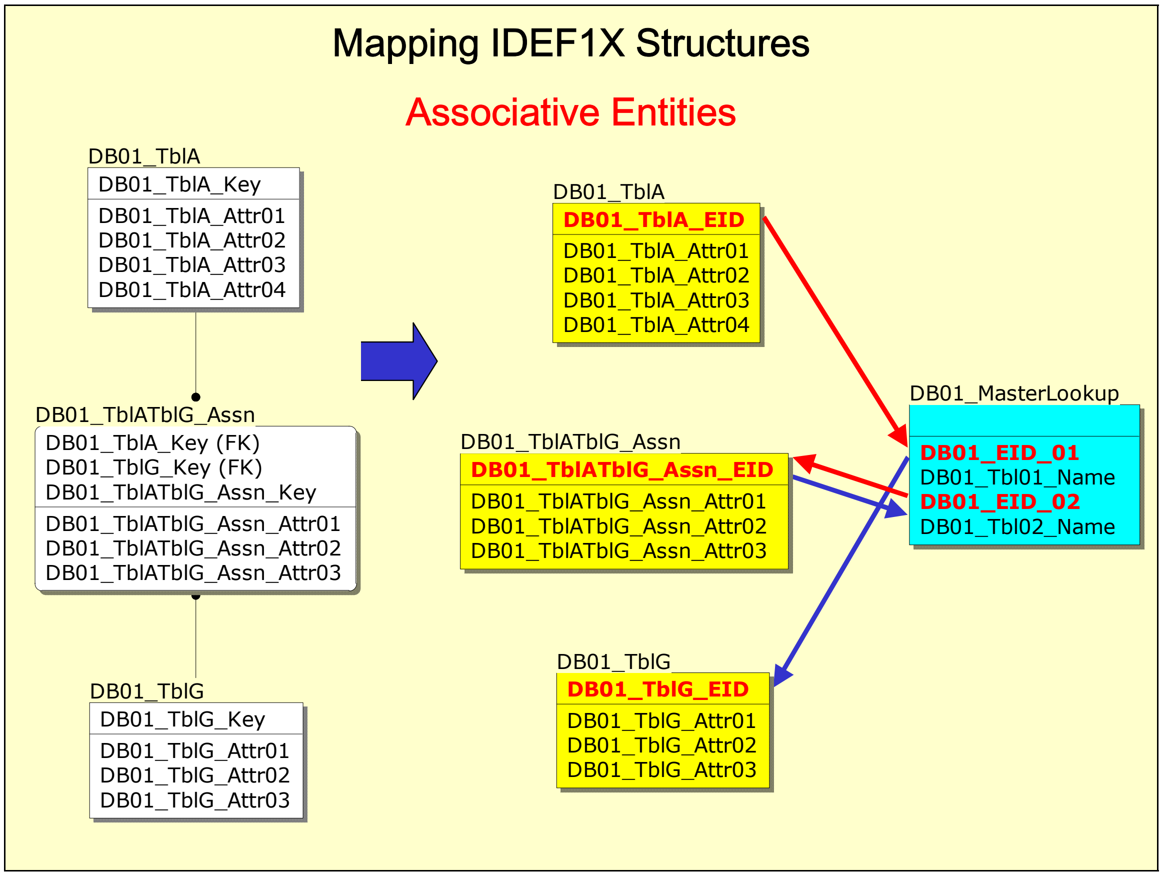


Figure A-27. Any Associative Entity Maps to a Record-Level Entry within the MasterLookup

[71] This may not be as unusual as one would initially think. For example, in the LC2IEDM some hierarchies contain multiple intermediate nodes whose sole purpose is to create nodes for the next level. These intermediate node entities tend to contain very little added information other than the category code that serves as a discriminant for the subtypes. With the use of EIDs discussed here, one could make the retrieval of data from the leaf much faster since there is no inherent advantage in going through the intermediate supertypes. Thus, for example, in a given situation records in OBJECT-ITEM could be directly linked to the appropriate records in UNIT, BRIDGE, etc., without need to go through ORGANISATION or FACILITY.

Figure A-27 depicts the mapping of an associative entity. An association represents a many-to-many relationship between two entities (therefore also called a many-to-many breaker), in this case, DB01_TblA and DB01_TblG. This relationship is resolved as a pair of one-to-many relationships between the parent entities, DB01_TblA and DB01_TblG, and the child associative entity, DB01_TblATblG_Assn. Indeed, the mapping of this type of data structure to an EID-based system consists of recording exactly two parent-child relationships of the kind that has been discussed before.

The right hand side of Figure A-27 shows the full tracing via the DB01_MasterLookup table between the two parent entities and the associative entity. Each record in DB01_TblATblG_Assn contains two entries in DB01_MasterLookup table. The first entry represents the relationship between DB01_TblA and DB01_TblATblG_Assn, and the second entry represents the relationship between DB01_TblG and DB01_TblATblG_Assn. Following the red arrows representing the first entry, we are able to extract the EID of DB01_TblATblG_Assn. Then, using the second entry marked by blue arrows, we can complete the path to DB01_TblG. Of course, the tracing of the paths is completely flexible: backwards from DB01_TblG to DB01_TblA, or from the center, i.e., out from DB01_TblATblG_Assn to DB01_TblA and DB01_TblG.

Implementation of Double-Associative Relations Via EIDs and the MasterLookup Table

Alternative 1

The double-association is a special extension of the examples previously described. In a simple associative relationship, such as the one schematically depicted in Figure A-27 above, the ordinate and subordinate entities are easily identified because the associative entity DB01_TblATblG_Assn links two different entities, namely DB01_TblA and DB01_TblG.

However, in a double-association, this is clearly not true, i.e., the two parents coalesce into one. Therefore, in the IDEF1X model, the subordination information, i.e., which record corresponds to the parent and which record corresponds to the child, is captured by role-naming the migrated keys, namely, DB01_Ordinate_TblA_Key(FK) for the parent record and DB01_Subordinate_TblA_Key(FK) for the child record (see the IDEF1X diagram on the left side of Figure A-28).

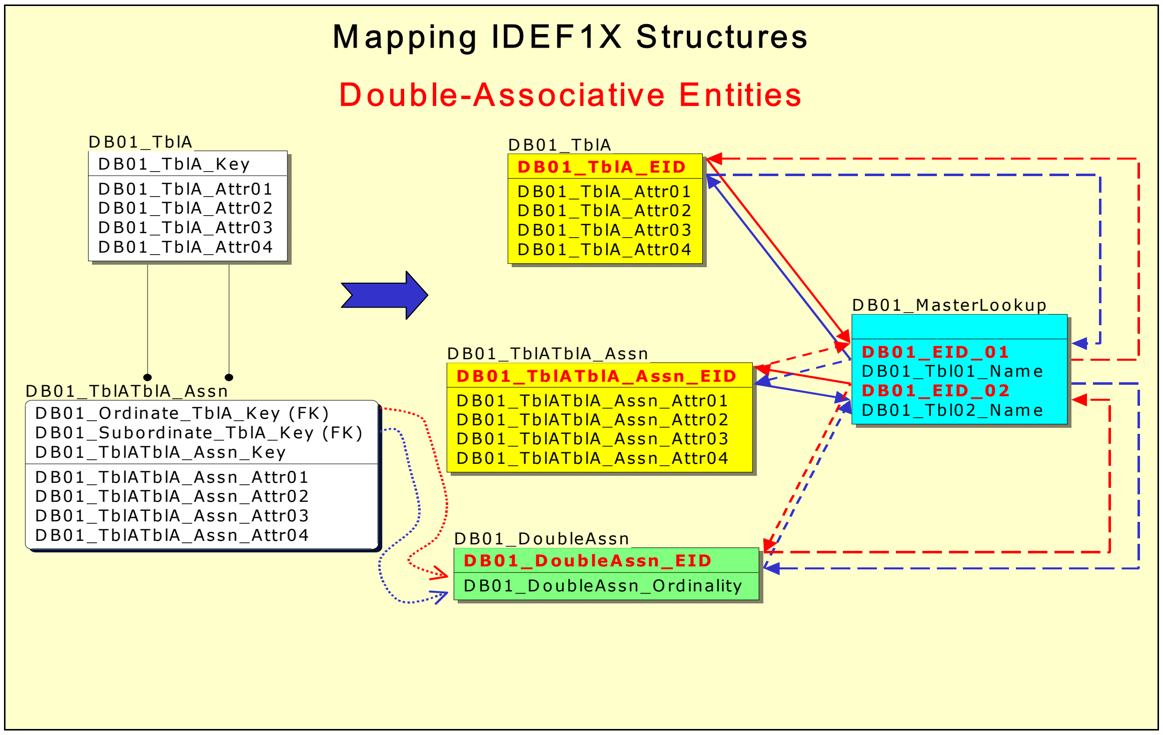


Figure A-28. Any Double Associative Entity Maps to a Record-Level Entry within the MasterLookup

When one introduces EIDs for all records and removes the traditional relationships between tables, however, the information contained in the role-naming key does not port over, since the EIDs carry no meaning or information whatsoever. Therefore, one must recreate this information somehow. The mapping shown on the right side of Figure A-28 is only one of many possible solutions to resolve the ambiguity of the EID roles, i.e., which one ought to be seen as the EID of the parent record and which one is the corresponding EID for the child record.

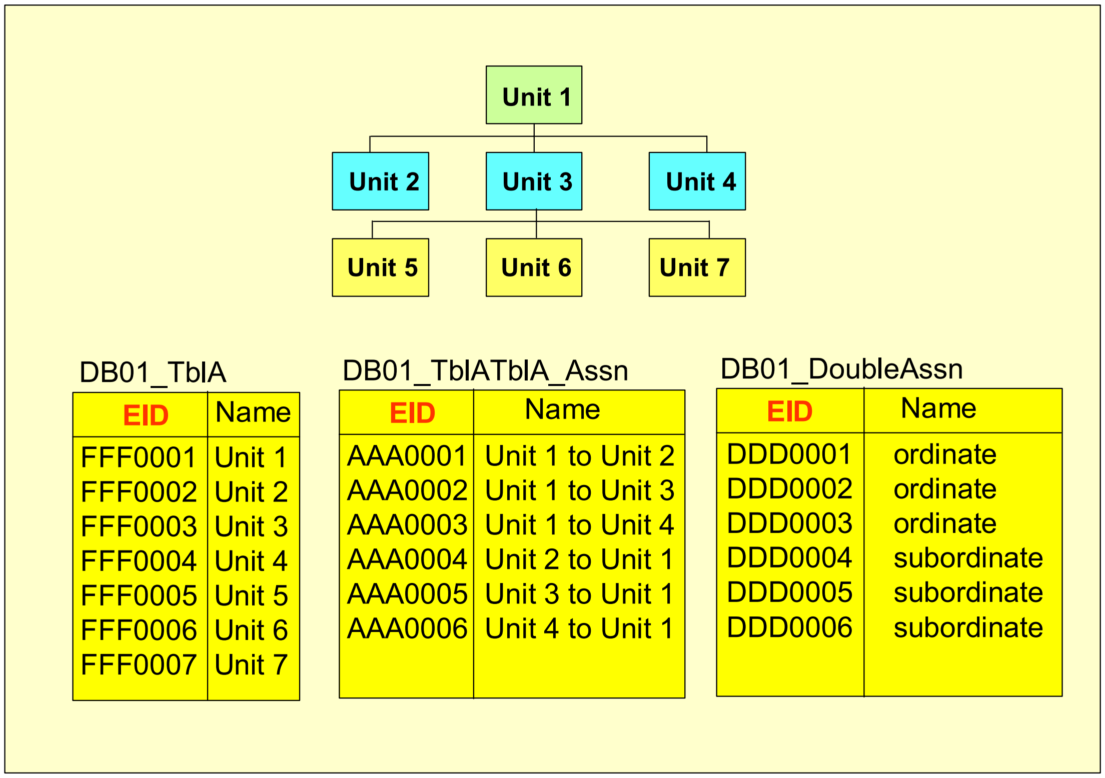
The method shown here requires the creation of a new entity, which is shown notionally as DB01_DoubleAssn in Figure A-28. It is a very simple table that records which EID is the ordinate EID and which one is the subordinate EID. The complete mapping is represented as three pairs of entries in the DB01_MasterLookup table: between DB01_TblA and DB01_TblATblA_Assn (solid red and blue arrows), between DB01_TblATblA_Assn and DB01_DoubleAssn (dotted red and blue arrows), and between DB01_TblA and DB01_DoubleAssn (dashed red and blue arrows).

Thus, to find all the instances in DB01_TblA that are subordinate to a given entry in DB01_TblA, one would query the DB01_MasterLookup table for all the EID values of DB01_TblATblA_Assn linked to that instance. Those EID values of DB01_TblATblA_Assn would in turn point to all the associated instances of DB01_DoubleAssn and show the role for that association. Similarly, the EID values for the instances in DB01_TblA also are related to a role recorded in DB01_DoubleAssn. Therefore, once the EID for the appropriate role—in our example, 'subordinate'—has been found, it can be used to retrieve the instances in DB01_TblA for which the initial record is the 'ordinate'.

These entries completely exhaust all the search paths, enabling queries in either direction and from any point. The number of entries in DB01_MasterLookup can be relaxed to four if the search is always performed in one direction, i.e., either from ordinate to subordinate or vice versa.

Alternative 2

The discussion under Alternative 1 above assumes that one creates only a single record in DB01_TblATblA_Assn per association between instances of the parent entity DB01_TblA. Thus two instances in DB01_TblA, namely, instance a and instance b, which stand in an ordinate-subordinate relationship captured in DB01_TblATblA_Assn as instance c, would show up in the DB01_MasterLookup as two pairs (instance a, instance c) and (instance b, instance c). And, as was stated above, under this circumstances there is no way to figure out whether instance a is the ordinate or the subordinate in the association. Through the use of DB01_DoubleAssn, one can disambiguate the roles of instance a and instance b.



DB01_TblA

EID	Name
FFF0001	Unit 1
FFF0002	Unit 2
FFF0003	Unit 3
FFF0004	Unit 4
FFF0005	Unit 5
FFF0006	Unit 6
FFF0007	Unit 7

DB01_TblATblA_Assn

EID	Name
AAA0001	Unit 1 to Unit 2
AAA0002	Unit 1 to Unit 3
AAA0003	Unit 1 to Unit 4
AAA0004	Unit 2 to Unit 1
AAA0005	Unit 3 to Unit 1
AAA0006	Unit 4 to Unit 1

DB01_DoubleAssn

EID	Name
DDD0001	ordinate
DDD0002	ordinate
DDD0003	ordinate
DDD0004	subordinate
DDD0005	subordinate
DDD0006	subordinate

Figure A-29. Alternative Approach for Handling a Double-Association via MasterLookup

A slightly different approach would be to create two instances in DB01_TblATblA_Assn, namely, one for the relation instance a *is ordinate for* instance b, and another for the relation instance b *is subordinate to* instance a. The entries in DB01_DoubleAssn would still be needed to label the role of the instances of DB01_TblA and would act as a filter, i.e., to prevent that the parent record be shown as subordinate to the child record.

However, this approach may permit a simplified logic to discern which records in DB01_TblA are subordinate to another instance in DB01_TblA, since there would be no need to relate the entries in DB01_TblA to the entries in DB01_DoubleAssn as is done under Alternative 1 above. Figure A-29 above shows a simple command structure among 7 notional units. If one were to record as one entry in the DB01_TblATblA_Assn the instance where the relation is read from ordinate to subordinate, and as a different entry when the relation is read from subordinate to ordinate, then for the first group of entities in Figure A-29, namely, Unit 1 through 4, there would be a total of 6 entries.

DB01_MasterLookup

EID_01	DB01_Tbl01_Name	EID_02	DB01_Tbl02_Name
FFF0001	DB01_TblA	AAA0001	DB01_TblATblA_Assn
FFF0001	DB01_TblA	AAA0002	DB01_TblATblA_Assn
FFF0001	DB01_TblA	AAA0003	DB01_TblATblA_Assn
FFF0002	DB01_TblA	AAA0004	DB01_TblATblA_Assn
FFF0003	DB01_TblA	AAA0005	DB01_TblATblA_Assn
FFF0004	DB01_TblA	AAA0006	DB01_TblATblA_Assn
AAA0001	DB01_TblATblA_Assn	FFF0002	DB01_TblA
AAA0002	DB01_TblATblA_Assn	FFF0003	DB01_TblA
AAA0003	DB01_TblATblA_Assn	FFF0004	DB01_TblA
AAA0004	DB01_TblATblA_Assn	FFF0001	DB01_TblA
AAA0005	DB01_TblATblA_Assn	FFF0001	DB01_TblA
AAA0006	DB01_TblATblA_Assn	FFF0001	DB01_TblA
AAA0001	DB01_TblATblA_Assn	DDD0001	DB01_DoubleAssn
AAA0002	DB01_TblATblA_Assn	DDD0002	DB01_DoubleAssn
AAA0003	DB01_TblATblA_Assn	DDD0003	DB01_DoubleAssn
AAA0004	DB01_TblATblA_Assn	DDD0004	DB01_DoubleAssn
AAA0005	DB01_TblATblA_Assn	DDD0005	DB01_DoubleAssn
AAA0006	DB01_TblATblA_Assn	DDD0006	DB01_DoubleAssn

Let us assume that one reads the information contained in DB01_DoubleAssn as expressing the role of the record linked to its corresponding instance in DB01_TblATblA_Assn. Then the sample entries in DB01_MasterLookup above could be used to find all the subordinate Units of Unit 1 by retrieving first all the EID values linked to the EID FFF0001 where the DB01_Tbl01_Name is equal to DB01_TblA and the

DB01_Tbl02_Name is equal to DB01_TblATblA_Assn. The values retrieved, namely AAA0001, AAA0002, and AAA0003, could then be readily used to check what kind of role the record with EID equal to FFF0001 was playing in that association. One could simply retrieve DDD0001, DDD0002, and DDD0003 and obtain the values of the role attribute in DB01_DoubleAssn—which in this case would all be 'ordinate'. The same values, namely AAA0001, AAA0002, and AAA0003 from DB01_TblATblA_Assn, can also be used to query back the records related to it in DB01_TblA. Doing this in the above example would retrieve FFF0002, FFF0003, and FFF0004, which are the EID values of Unit 2, Unit 3 and Unit 4, respectively.

If instead of querying DB01with the EID value of Unit 1, one queries it with an EID value equal to FFF0004, then the DB01_MasterLookup would show only one entry for DB01_TblATblA_Assn namely, AAA0006. This entry is related to the entry with EID DDD0006 in the table DB01_DoubleAssn with role 'subordinate', and to the entry with EID value FFF0001 in DB01_TblA. This would mean that Unit 4 stands in a subordinate relationship to Unit 1, and, furthermore, that it does not have any more relationships either as ordinate or as subordinate to any other entry in DB01_TblA.

Alternative 3

The multiple passes necessitated by either Alternative 1 or Alternative 2 are the result of making the DB01_MasterLookup as simple as possible, namely, just four attributes and nothing else. For the case of double associative entities, one could consider the addition of another attribute. This attribute would capture the role of a given instance of DB01_TblA in the association table DB01_TblATblA_Assn, therewith bypassing completely the need for the DB01_DoubleAssn table.

Since in many cases the majority of the data resident in databases is not in double associative entities, this would mean that the bulk of the entries in the DB01_MasterLookup would have no value assigned for the role attribute. Though wasteful, this approach, coupled with the idea of assigning a different EID for the associations according to the direction in which they are read, may significantly reduce query times when traversing a large DB01_MasterLookup in search for information such as force structure, materiel components listed as subcomponents of larger assemblies, etc.[72]

[72] At the time of writing this analysis, only Alternative 1 has been tested in an information system using LC2IEDM, as the physical schema implemented in SQL Server 2000, and a Web-based interface controlling the logic for access to the data.

Illustrative Example Using EIDs and the MasterLookup Table in LC2IDEM: Unit Holding

In the LC2IEDM, Holding can be used to represent the on-hand inventory of a military unit. For example, using the Holding table, we can find out the type and quantity of a class of materiel in a unit's inventory: small arms, ammunition, food supplies, fuel, vehicles, etc. We can also use Holding to find out the type and quantity of personnel in the unit by rank and specialty.

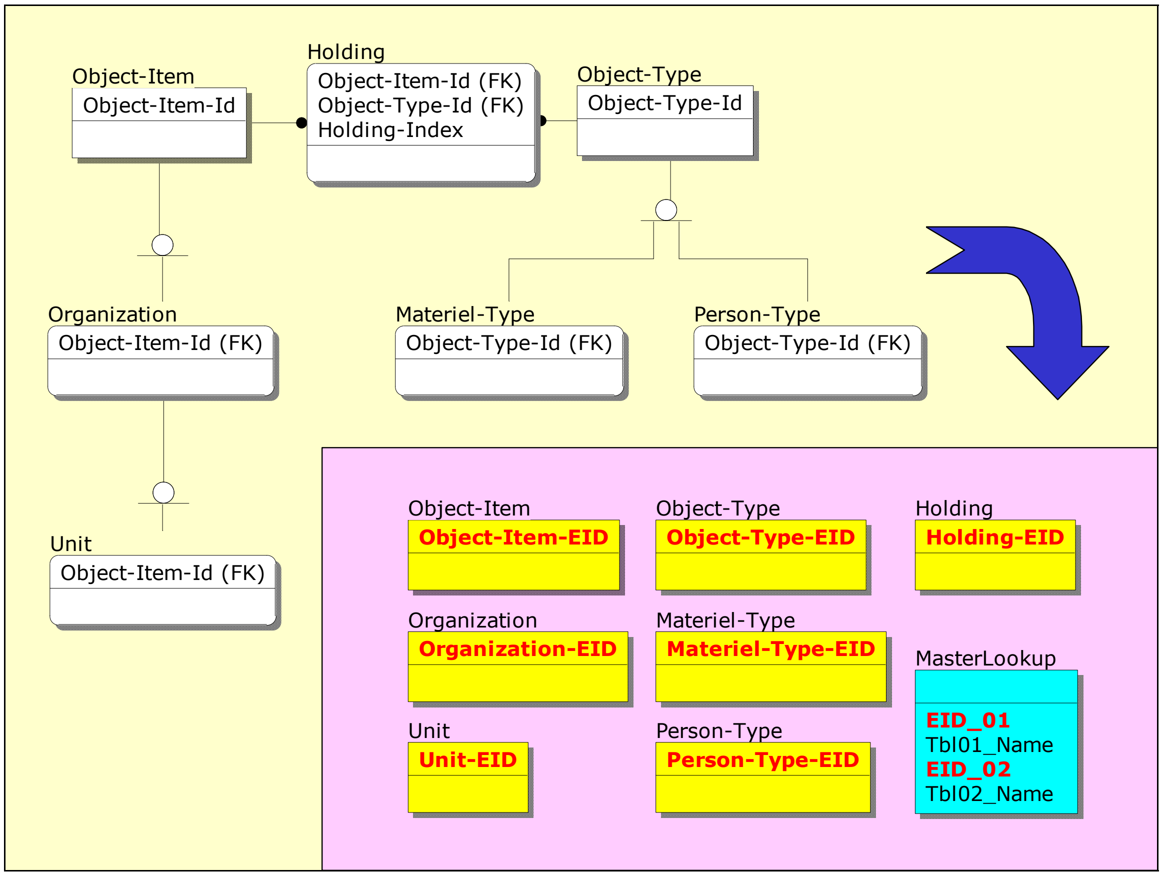
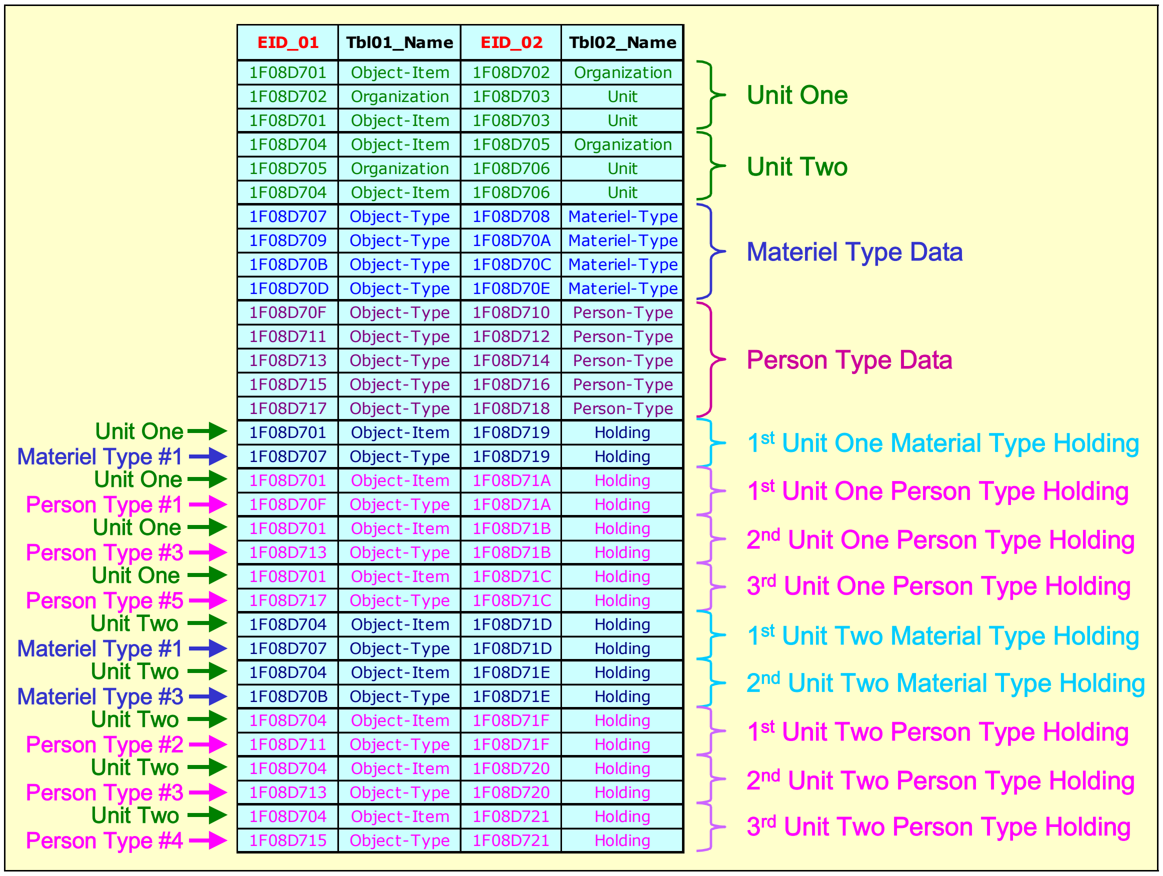


Figure A-30. Mapping of LC2IEDM Holding using EID and MasterLookup

Figure A-30 shows the IDEF1X diagram for the portion of LC2IEDM that includes Holding. As shown in the upper part of Figure A-30, the entity Holding is an associative entity that contains data on the quantity of a particular Object-Type in the inventory of an Object-Item—in this case a military unit. Note that Units are part of the Object-Item hierarchy, and, therefore, we can retrieve the Unit information represented by this generalization. Similarly, through Object-Type, we can retrieve the classes of materiel and types of personnel the unit has through the Object-Type hierarchy that contains both Materiel-Type and Person-Type. Converting this type of entity-relationship structure to an EID-based structure with lookup table is straightforward. The lower part of Figure A-30 shows how the LC2IEDM portion for Holding would be represented. As discussed in the preceding paragraphs, making the EID the primary key for all entities transforms them into independent ones. All records are now identified by globally unique EIDs. All logical links among the entities are recorded in the MasterLookup table (see lower right hand corner of Figure A-30).

Figure A-31 below shows a portion of the MasterLookup table where the various types of relationships discussed previously in this section are captured through the entries therein. The Object-Item subtype hierarchy represents a generalization of multiple levels. Its mapping to the new structure corresponds to a cascading relationship (see Figure A-26 above). The Object-Type hierarchy is an example of multiple subtypes on a single level. This is mapped simply to parent-child relationships with Z-cardinality (see also Figure A-24 above). Lastly, the Holding is a normal association, and its mapping corresponds to two parent-child relationships as shown in Figure A-27 above.



Unit One
Unit Two
Materiel Type Data
Person Type Data
1st Unit One Material Type Holding
1st Unit One Person Type Holding
2nd Unit One Person Type Holding
3rd Unit One Person Type Holding
1st Unit Two Material Type Holding
2nd Unit Two Material Type Holding
1st Unit Two Person Type Holding
2nd Unit Two Person Type Holding
3rd Unit Two Person Type Holding
Unit One
Materiel Type #1
Unit One
Person Type #1
Unit One
Person Type #3
Unit One
Person Type #5
Unit Two
Materiel Type #1
Unit Two
Materiel Type #3
Unit Two
Person Type #2
Unit Two
Person Type #3
Unit Two
Person Type #4

EID_01	Tbl01_Name	EID_02	Tbl02_Name
1F08D701	Object-Item	1F08D702	Organization
1F08D702	Organization	1F08D703	Unit
1F08D701	Object-Item	1F08D703	Unit
1F08D704	Object-Item	1F08D705	Organization
1F08D705	Organization	1F08D706	Unit
1F08D704	Object-Item	1F08D706	Unit
1F08D707	Object-Type	1F08D708	Materiel-Type
1F08D709	Object-Type	1F08D70A	Materiel-Type
1F08D70B	Object-Type	1F08D70C	Materiel-Type
1F08D70D	Object-Type	1F08D70E	Materiel-Type
1F08D70F	Object-Type	1F08D710	Person-Type
1F08D711	Object-Type	1F08D712	Person-Type
1F08D713	Object-Type	1F08D714	Person-Type
1F08D715	Object-Type	1F08D716	Person-Type
1F08D717	Object-Type	1F08D718	Person-Type
1F08D701	Object-Item	1F08D719	Holding
1F08D707	Object-Type	1F08D719	Holding
1F08D701	Object-Item	1F08D71A	Holding
1F08D70F	Object-Type	1F08D71A	Holding
1F08D701	Object-Item	1F08D71B	Holding
1F08D713	Object-Type	1F08D71B	Holding
1F08D701	Object-Item	1F08D71C	Holding
1F08D717	Object-Type	1F08D71C	Holding
1F08D704	Object-Item	1F08D71D	Holding
1F08D707	Object-Type	1F08D71D	Holding
1F08D704	Object-Item	1F08D71E	Holding
1F08D70B	Object-Type	1F08D71E	Holding
1F08D704	Object-Item	1F08D71F	Holding
1F08D711	Object-Type	1F08D71F	Holding
1F08D704	Object-Item	1F08D720	Holding
1F08D713	Object-Type	1F08D720	Holding
1F08D704	Object-Item	1F08D721	Holding
1F08D715	Object-Type	1F08D721	Holding

Figure A-31. Example of MasterLookup entries for Unit Holding

The green entries correspond to the Object-Item subtype hierarchy. For example, the first three lines contain the entries for "Unit One." The first line is the first level of the hierarchy which reads that Organization 1F08D702 is the subtype of Object-Item 1F08D701. The second line is the second level of the hierarchy which reads that Unit 1F08D703 is the subtype of Organization 1F08D702. The third line represents an optional relationship that directly links Object-Item to Unit which reads that Unit 1F08D703 is the subtype of Object-Item 1F08D701. The entries for "Unit Two" follow in the exact same manner.

The next group of entries in blue are the entries for the Materiel-Type branch of the Object-Type hierarchy. For example, the first line simply says that Materiel-Type 1F08D708 is the subtype of Object-Type 1F08D707.

The next group of entries in purple are the entries for Person-Type, i.e., the second branch of the Object-Type hierarchy shown in Figure A-30 above. For example, the first line simply says that Person-Type 1F08D710 is the subtype of Object-Type 1F08D70F.

Finally, the rest of the entries, which occur in pairs, are the associations. Let us take the entry labeled "2nd Unit Two Materiel Type Holding" in Figure A-29 for detailed examination. The first line of the pair reads that Object-Item 1F08D704 has the Holding 1F08D71E. The second line reads that Object-Type 1F08D70B is the subject of Holding 1F08D71E. Using the entries for the Object-Item subtype hierarchy, we can see Unit 1F08D706 is the subtype of Object-Item 1F08D704 (from line 6). Furthermore, using the entries for the Object-Type hierarchy, we see that Materiel-Type #3 1F08D70C is the subtype of Object-Type 1F08D70B (from line 9). Therefore, we can conclude that "Unit Two" has in its inventory "Materiel Type #3," the quantity of which is recorded in the associated Holding.

Using the same process, the other pairs of Holding entries can be resolved to identify the full inventory, both personnel and materiel, for both "Unit One" and "Unit Two."

ILLUSTRATIVE EXAMPLE USING EIDS AND THE MASTERLOOKUP TABLE IN LC2IDEM: ORGANIZATIONAL HIERARCHY

The example discussed in Figure A-30 shows the required mappings for all the IDEF1X structures, except the one needed to represent a double association. Figure A-32 shows a good example for the use of a double association. The identification of organizational hierarchy (e.g., force structure, chain of command) is vital for C2 purposes. The LC2IEDM entity used to record this type of organization to organization relationships, such as their reporting hierarchy, is the Organization-Organization-Association.

The IDEF1X diagram in Figure A-31 shows the portion of the LC2IEDM that contains Org-Org-Assn. Once again, we see the Object-Item subtype hierarchy used in the identification of Unit that we used in the previous example. The double association, Org-Org-Assn, is an associative entity that has Org as both the subject and the object. Therefore, in order to distinguish the subject and the object, the migrated foreign keys have been role-named for identification. The mapping to an EID-based structure with a lookup table requires an additional entity to record the subject/object role. This is shown in Figure A-32 with the entity highlighted in green. As the previous example in Figure A-30 shows, with the introduction of EIDs as the primary keys, all the entities become independent and their records are uniquely identified by EIDs alone. The relationship between entities is recorded in MasterLookup, and any role information for a double association is recorded in DoubleAssn.

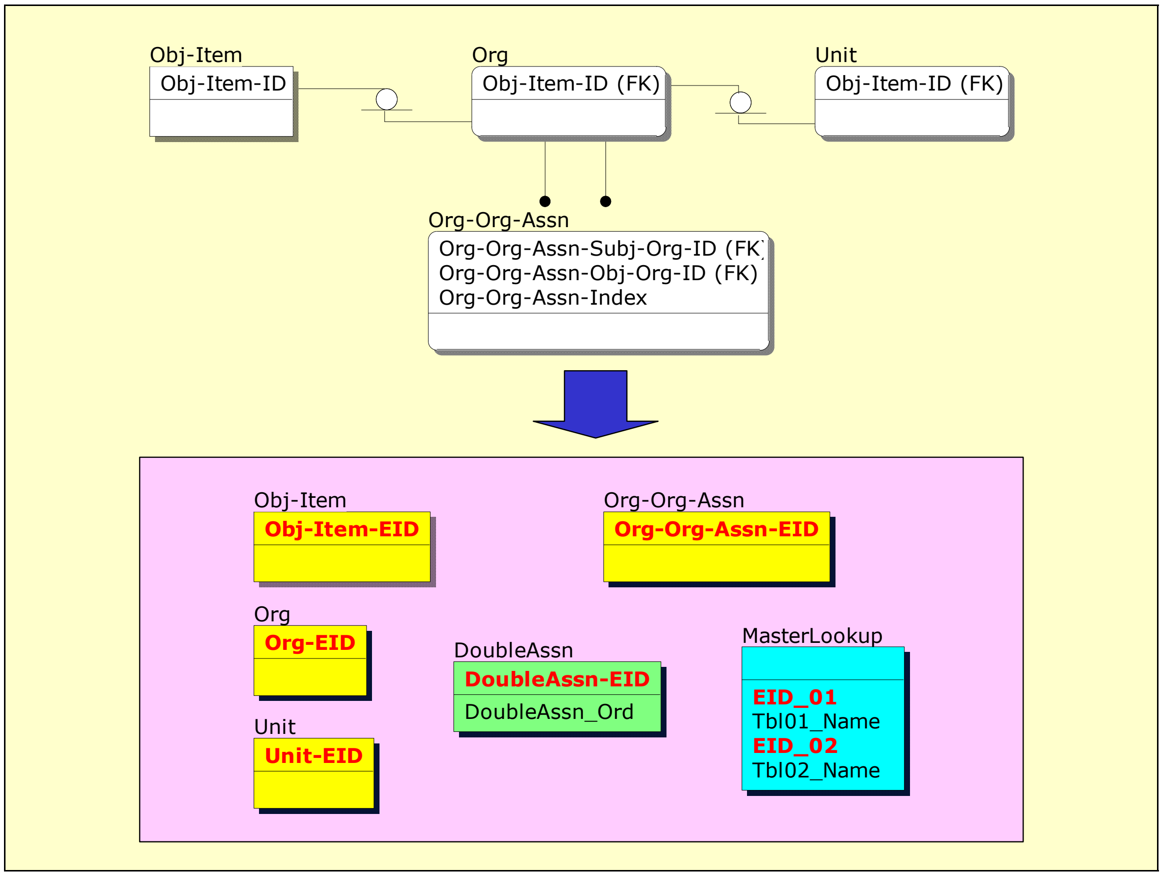
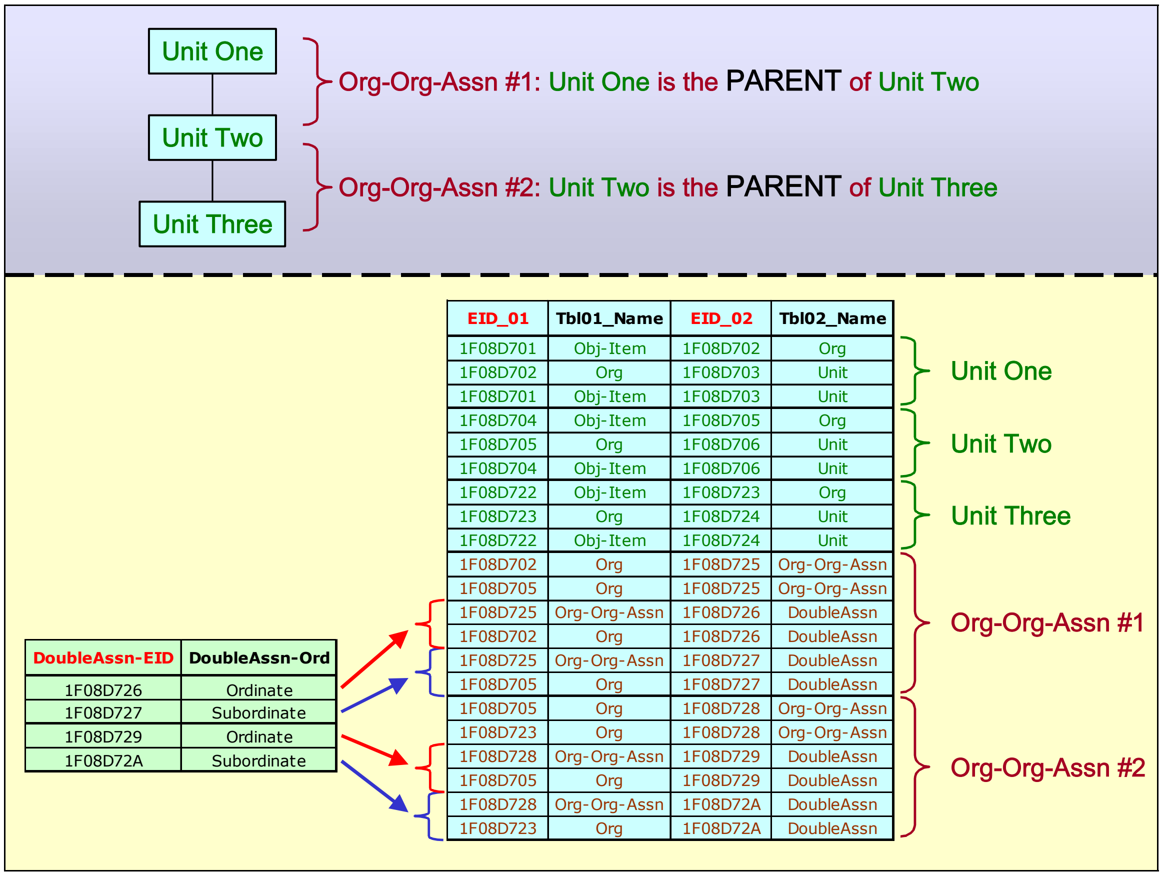


Figure A-32. Mapping of LC2IEDM Organization-Organization-Association using EID and MasterLookup

Figure A-33 shows the organizational hierarchy we want to capture and the corresponding entries in MasterLookup. This is a simple example where “Unit One” is the parent or commanding unit of “Unit Two” (represented by Org-Org-Assn #1), and “Unit Two” is the parent or commanding unit of “Unit Three” (represented by Org-Org-Assn #2). The green entries in the MasterLookup table correspond to the Object-Item subtype hierarchy that captures Unit information. The rest of the entries, colored in brown, record the double association, which occurs in groups of six.

Let us take a detailed examination of the first set of these entries for Org-Org-Assn #1. The first line reads that Org 1F08D702 has Org-Org-Assn 1F08D725. The second line reads that Org 1F08D705 also has Org-Org-Assn 1F08D725. Notice that, so far, the role cannot be inferred. The next pair of entries says that Org-Org-Assn 1F08D725 has a "role" record DoubleAssn 1F08D726 and that Org 1F08D702 also has a "role" record DoubleAssn 1F08D726. Tracing back to DoubleAssn 1F08D726, we find that the "role" is recorded as “Ordinate.” Therefore, we can now infer that Org 1F08D702 is the ordinate record for Org-Org-Assn 1F08D725. If we stop at this point with four entries in MasterLookup, we have successfully captured the double association relationship in the ordinate-to-subordinate direction. However, if we include the next pair, then the relationship is recorded in both the ordinate-to-subordinate and subordinate-to-ordinate direction. This pair says that Org-Org-Assn 1F08D725 has a "role" record DoubleAssn 1F08D727 and that Org 1F08D705 also has a "role" record DoubleAssn 1F08D727. Tracing back to DoubleAssn 1F08D727, we find that the "role" is recorded as “Subordinate.” Therefore, Org 1F08D705 is the subordinate record for Org-Org-Assn 1F08D725.



Unit One
Unit Two
Unit Three
Org-Org-Assn #1: Unit One is the PARENT of Unit Two
Org-Org-Assn #2: Unit Two is the PARENT of Unit Three
Unit One
Unit Two
Unit Three
Org-Org-Assn #1
Org-Org-Assn #2

EID_01	Tbl01_Name	EID_02	Tbl02_Name
1F08D701	Obj-Item	1F08D702	Org
1F08D702	Org	1F08D703	Unit
1F08D701	Obj-Item	1F08D703	Unit
1F08D704	Obj-Item	1F08D705	Org
1F08D705	Org	1F08D706	Unit
1F08D704	Obj-Item	1F08D706	Unit
1F08D722	Obj-Item	1F08D723	Org
1F08D723	Org	1F08D724	Unit
1F08D722	Obj-Item	1F08D724	Unit
1F08D702	Org	1F08D725	Org-Org-Assn
1F08D705	Org	1F08D725	Org-Org-Assn
1F08D725	Org-Org-Assn	1F08D726	DoubleAssn
1F08D702	Org	1F08D726	DoubleAssn
1F08D725	Org-Org-Assn	1F08D727	DoubleAssn
1F08D705	Org	1F08D727	DoubleAssn
1F08D705	Org	1F08D728	Org-Org-Assn
1F08D723	Org	1F08D728	Org-Org-Assn
1F08D728	Org-Org-Assn	1F08D729	DoubleAssn
1F08D705	Org	1F08D729	DoubleAssn
1F08D728	Org-Org-Assn	1F08D72A	DoubleAssn
1F08D723	Org	1F08D72A	DoubleAssn

DoubleAssn-EID	DoubleAssn-Ord
1F08D726	Ordinate
1F08D727	Subordinate
1F08D729	Ordinate
1F08D72A	Subordinate

Figure A-33. Example of MasterLookup entries for Organizational Hierarchy

Finally, taking all the information together, we can say that Org 1F08D702, which is Unit 1F08D703 (second line from the top) or “Unit One,” is the parent organization to Org 1F08D705, which is Unit 1F08D706 (fifth line from the top) or “Unit Two.” Furthermore, this relationship is subject to the conditions set forth in Org-Org-Assn 1F08D725. In LC2IEDM different Org-Org-Assn records can be used to identify different types of organization to organization relationships, as encoded by the enumerated domains of the category and subcategory codes. For example, one set of records may be used to capture the administrative hierarchy while another set is used to capture the command hierarchy; one set may be used to capture the present command structure while another set can be used for recording a planned modification schedule to take place in the next fiscal year.

D. PERFORMANCE CONSIDERATIONS WHEN USING ORG IDs IN NEW SYSTEMS

When using EIDs such as ORG ID for record identification in participating information systems within the DoD enterprise, one of the most important criteria is response time for operations dependent on access to records tagged with such identifiers.

Performance degradation may be due to the following:

- structure of the EIDs themselves
- choice of implementation of the record identifier within a particular RDBMS—either as primary key or as alternate key

With respect to the first possible impact of using EIDs as the record identifier, the following alternatives can be explored:

- if performance degradation arises from a lack of native SQL data types in the RDBMS to store the EID, then either indexing or hash tables may be appropriate.

For example, choosing EIDs as 64-bit long integers may not be supported by older RDBMSs, such as Microsoft SQL 7.0. Desktop database applications such as Microsoft Access may record the EIDs using some other form of internal representation, e.g., the currency data type.

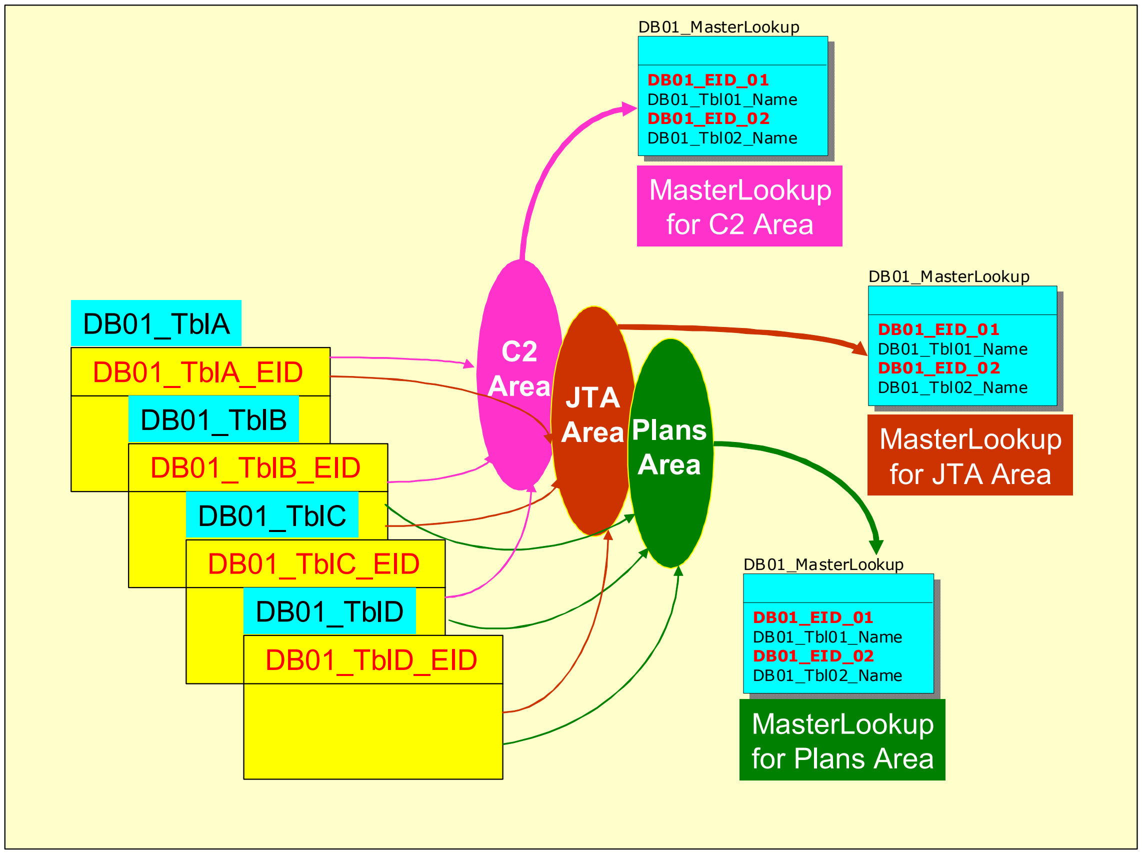
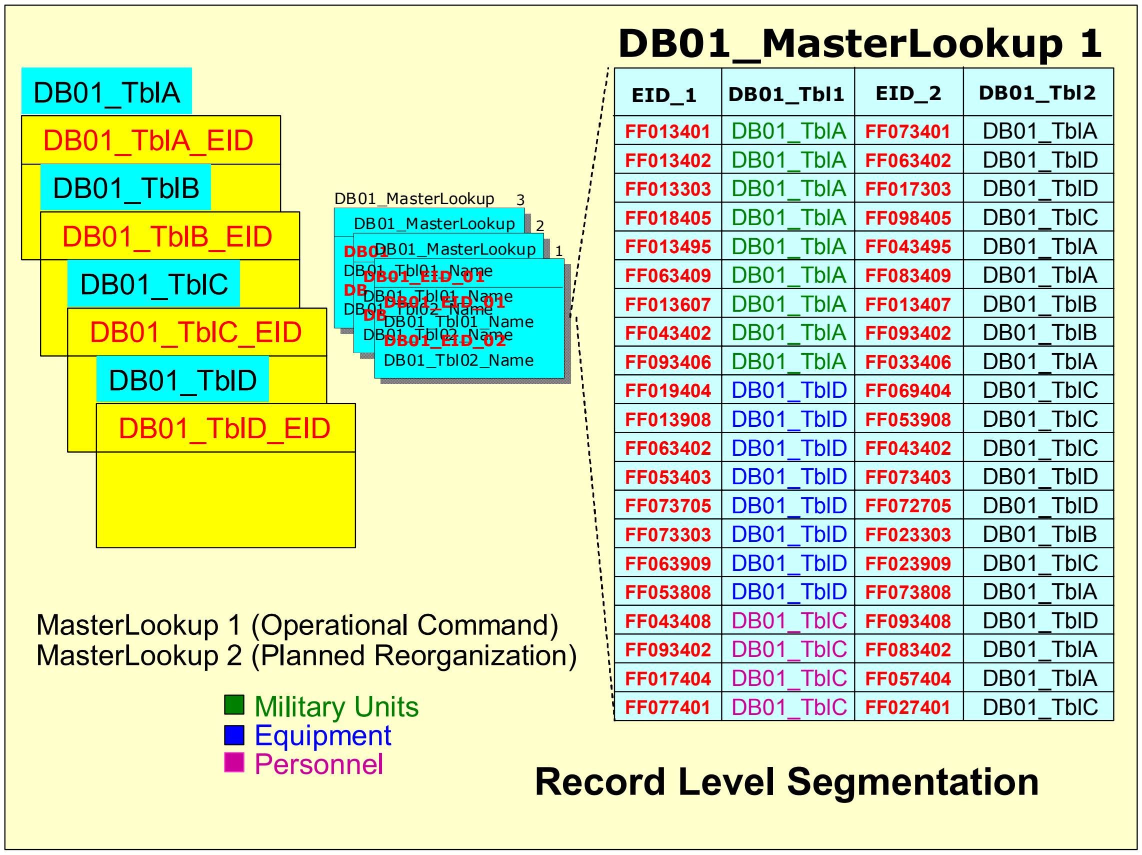


Figure A-34. Segmentation of the MasterLookup by Area as a Means to Prevent Performance Degradation

If EIDs are implemented with commercial products, such as Microsoft database Replication Identifier (GUID) which is a 128 bit long integer, some databases may be able to record the EIDs only as the ASCII string corresponding to the Hexadecimal representation of the GUID values. In such cases, queries with large data sets may be less efficient than if the EIDs were natively stored as numeric values. Indexing or hash tables may be an alternative to ameliorate the problem of unacceptably long response times.

A second possible source of performance degradation is the use of EIDs as alternate keys or primary keys in conjunction with the regular structuring of relationships—identifying, non-identifying and supertype-subtype hierarchies. However, no special performance degradation should occur because the keys used are EIDs rather than keys unique at the table level only. If relationships are captured at the record level rather than at the table level, then the following two alternatives can be explored to prevent performance degradation:

- segmentation of the data encoding the record-level relationships by area.



EID_1	DB01_Tbl1	EID_2	DB01_Tbl2
FF013401	DB01_TblA	FF073401	DB01_TblA
FF013402	DB01_TblA	FF063402	DB01_TblD
FF013303	DB01_TblA	FF017303	DB01_TblD
FF018405	DB01_TblA	FF098405	DB01_TblC
FF013495	DB01_TblA	FF043495	DB01_TblA
FF063409	DB01_TblA	FF083409	DB01_TblA
FF013607	DB01_TblA	FF013407	DB01_TblB
FF043402	DB01_TblA	FF093402	DB01_TblB
FF093406	DB01_TblA	FF033406	DB01_TblA
FF019404	DB01_TblD	FF069404	DB01_TblC
FF013908	DB01_TblD	FF053908	DB01_TblC
FF063402	DB01_TblD	FF043402	DB01_TblC
FF053403	DB01_TblD	FF073403	DB01_TblD
FF073705	DB01_TblD	FF072705	DB01_TblD
FF073303	DB01_TblD	FF023303	DB01_TblB
FF063909	DB01_TblD	FF023909	DB01_TblC
FF053808	DB01_TblD	FF073808	DB01_TblA
FF043408	DB01_TblC	FF093408	DB01_TblD
FF093402	DB01_TblC	FF083402	DB01_TblA
FF017404	DB01_TblC	FF057404	DB01_TblA
FF077401	DB01_TblC	FF027401	DB01_TblC

Figure A-35. Segmentation of the MasterLookup at Record-Level as a Means to Prevent Performance Degradation

Instead of one large lookup table where the data for all records reside, multiple lookup tables may be used to capture information pertinent to C2 operations. A different lookup table also may be used for planning data or technical architecture data. Figure A-34 above depicts how segmentation of the lookup table by area can ameliorate performance degradation in a given implementation.

The second approach that may be explored to prevent performance degradation is:

- segmentation of the data encoding the record-level relationships by record.

Again, instead of implementing a single large lookup table to relate the EIDs of the records related to each other in some form, multiple lookup tables may contain the EIDs for the relationships segmented according to specific criteria. Figure A-35 depicts the approach for record-level segmentation of the lookup table.

For example, if military units, equipment and personnel are related to other military units, equipment and personnel under operational command, and they are also related to each other under future reorganization plans, then the owner of the information system may maintain two different lookup tables.

The information system interface will search one or the other lookup table based on the entry point into the system by a user or another external system. For example, if the system is set up to provide information via the web, e.g., as a web portal, then the system knows which interface is initiating the query and code behind each page can be tailored so that the system automatically searches the appropriate lookup table. Additional measures may include dynamic resorting of the lookup table to rank relationships based on the frequency with which they are used.For instance, they may be placed at the beginning of the lookup table rather than reside by entry order within it.

E. CONSIDERATIONS ON LOADING EID VALUES INTO LEGACY SYSTEMS

The previous sections have shown the different alternatives for implementing EIDs such as ORG-IDs both in legacy as well as in new systems. The discussion shows that EIDs can be introduced quite readily in legacy databases as an alternate key with minimal or no impact to the information system. Basically all that is required is to run an SQL script against the physical schema implemented in a given RDBMS that alters every table and inserts a new attribute called EID.[73] Modifications to legacy systems in which the EID is used only as the primary key of independent entities is slightly more complex. But since the number of such entities generally tends to be small, it is probably not something requiring inordinate amounts of effort and time.[74]

Assuming the above statements are basically correct, it becomes clear that the issue is no longer the modification to the schema, either in terms of additional attributes or in terms of key structure changes, but the loading of the legacy data into a modified target system that has adopted EIDs.

Case 1.

If the modification to the database physical schema consists in the addition of an EID attribute to the RDBMS tables that can act as an alternate key, then the assignment of EIDs can be conducted also by, for example, a simple SQL script performing a series of `INSERT VALUES` operations for each table. Here again the mechanics of the process present no major difficulties.

Clearly, whether the EID is a 128 bit number generated by a commercial application such as Microsoft's GUID generator, or whether it is a 64 bit number created by first obtaining

---

[73] The simplest SQL command that accomplishes this has the general form `ALTER TABLE` **%TABLE NAME%** `ADD (`**%ATTRIBUTE% %SQL DATATYPE%**`);` For example, to add in an ORACLE database the attribute EID to a table called DB01_TblA with SQL data type set to NUMBER (20), one would execute the following SQL script:

```
ALTER TABLE DB01_TblA

ADD(EID NUMBER(20));
```

If the database has N tables, and all of them are meant to contain EIDs then one needs to execute N such statements. Setting other constraints for the EID attribute can also be accomplished by using more elaborate versions of the `ALTER TABLE` command, e.g., setting the NOT NULL property to make it mandatory, etc. Irrespective of the form chosen for the `ALTER TABLE` command, it is arguably a relatively straightforward operation that can be performed by the database administrator quite readily at the appropriate time—for example during the downtime scheduled for system maintenance.

Since in the case of insertion of EIDs in legacy systems as alternate keys the new EID attribute is not involved in any of the procedures stored in the database, this alteration of the physical schema of the database by itself carries minimal or no risk to the overall operation of the information system using the database as its data store.

[74] Most physical schemas are currently generated from logical data models maintained separately. In the case of substantial modifications to the database structure, it is probably more efficient to go back directly to the tool where the model is maintained, carry out the alterations to the structures, and then forward-engineer a new SQL script to create the instance of the database.

a centrally controlled 32 bit EID seed and concatenating it with a sequentially increasing second 32 bit that is locally managed, or whether the EID is produced by any other mechanism that ensures the global uniqueness of the EID, these EIDs would guarantee that every record in every database so modified possesses a record identifier that does not repeat elsewhere.

However, if the various information systems of the enterprise were to conduct this type of EID assignments without any coordination among themselves, then the overall benefits of adopting EIDs would be substantially reduced, since now we have gone from a semantic and syntactic stove-piping to something that looks like identifier stove-piping. In other words, even though the records now can be uniquely identified, System A could not readily reuse the data stored in System B via the EID mechanism because even if both systems are tracking, for example, the same classes of materiel, the EIDs in System A would be different from those used in System B. If System B queries System A using its own EIDs it would not be able to retrieve any data because System A does not know of the EIDs that System B is using, and vice versa.[75]

The preceding paragraphs clearly show that there is a need to institutionalize a series of 'authoritative data sources' managed by those agencies and organizations whose business is to create the new instances of, for example, materiel classes, occupational specialties, etc. Instead, all information systems in the DoD enterprise can use the same values of the EIDs for retrieving or sending data pertinent to those instances efficiently and without the risk of creating a semantic disconnect.

Case 2.

If the EIDs are now used as the primary keys in a legacy system as discussed in Section B.2 (see Figures A-13 through A-17), the impact to the RDBMS is more substantial. Any stored procedure based on the old keys—now treated as alternate keys—may be impacted. In addition, the insertion of values for the new keys requires more care, since EID values in parent entities must match the values loaded in child entities where they migrate as foreign keys. This avoids referential integrity problems when reloading legacy data into the modified physical schema.

But aside from the more time-consuming aspect of loading EID values that must satisfy referential integrity constraints, the potential for identifier stove-piping is also present here. Therefore, what was said in terms of the need for coordinating 'authoritative data sources' applies with equal force to Case 2 as it did for Case 1 discussed above.

[75] There is, of course, a great advantage in having at least the same structure for the EIDs, be it a 64 bit number or some other form, since, at a minimum, by adopting a standard SQL data type for the EID the databases would not have differing syntactic specifications for the same attribute and could make sense at least of that portion of the data exchanged. If in addition all systems were to adopt a *'publish and subscribe'* approach, then their data also could be available via automated queries based on EIDs from anywhere within the enterprise. This approach is not so easily implementable in the absence of a common *'record naming convention'*, which is the basic characteristic and benefit of using EIDs. However, it should be clear that a *'publish and subscribe'* approach does not prevent the problem alluded here of *'identifier stove-piping'*.

Technical Approaches to Avoid Identifier Stove-Piping when Populating EIDs Introduced Into A Legacy Database

Because identifier stove-piping is something that ought to be avoided whenever possible, the following sections will explore the potential of some techniques for automating the insertion of EID values into legacy systems. The discussion will assume that 'authoritative data sources' are available and that the participating information systems want to take advantage of these resources to integrate their services.

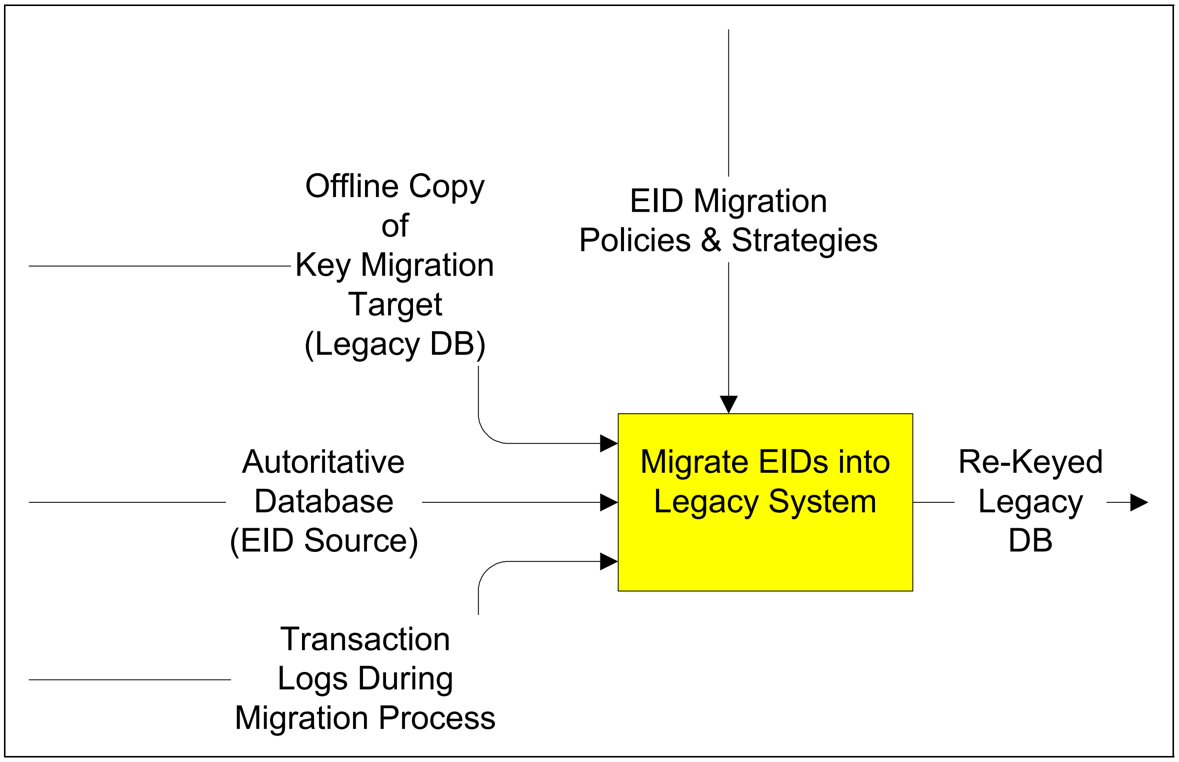


Figure A-36. A Process Context for Migrating EIDs into Legacy Databases

Figure A-36 schematically depicts the generic process that would be involved in the assignment of EID values to records resident in a legacy database. This assumes that these values match those of a pertinent authoritative data source, and, therefore, that they ought to use the same EID values for its records. Under this scenario, an offline copy of the legacy database, containing entities targeted to use EID values from an authoritative data source, would have to be matched to the appropriate entities from said authoritative data source to create a re-keyed version of the legacy database. Such a process is feasible if the targeted entities could be semantically associated with some corresponding entities in the authoritative database. In that case, a human (or an algorithm) could associate each record in the legacy table with a record in a source table. For each successful match, the value in the EID field of the authoritative data source would then be copied into the EID field of the legacy table.

The nature and extent of the processing involved depends on the desired role of the EIDs. The most complex level of processing would occur in a scenario where EIDs were to be installed as a new set of primary keys. In this case, the re-keyed database would first copy the EID values, resulting from record-by-record matches from authoritative data source, into the associated records of the target entities. Next, the EIDs would be propagated from the independent entities to the dependent entities. Then, the existing queries and user interfaces would be updated to reflect the new primary keys. Finally, the updated target database would be resynchronized with the on-line version to reflect transactions which occurred during the migration process, and brought on-line.

Figure A-35 shows, in addition to the inputs and outputs, a control input based on policies and strategies. These include the issue of which legacy entities will be targeted to receive EIDs. For instance, should the migration strategy be a phased migration in which only a few entities are targeted for migration at a time? If so, then before commencement of the next migration phase, the legacy system would be returned to operational status for a period of evaluation to better manage the risk associated with changes to the key structure. This migration strategy would avoid a change that could affect a large number of tables simultaneously.

The migration policies and strategies may also include a policy that ensures data confidentiality by preventing the targeting of certain tables in a given legacy system.[76] Another policy might establish a maximum allowable rate of incorrect EID assignments. This policy would recognize that human and automated EID assignments might not always be correct. Therefore, one should put in place standards that prevent the re-keying of target entities which have poor alignment with the data source.

For the purpose of this analysis of how to incorporate EID values from authoritative data sources into a particular legacy system, we will concentrate on the three major components of EID migration: (1) importing EIDs into a single independent entity; (2) propagating EIDs across parent-child relationships and supertype-subtype hierarchies as an alternate key; and (3) converting alternate keys into primary keys.

These tasks are interrelated since those activities involved in importing the EID values support (or are required by) the tasks of propagating them to the child entities and of promoting alternate keys to primary keys. In the following discussion, we will consider scenarios which illustrate the technical approach to executing these tasks.

[76] Personnel data likely would have to be managed in a way that complies with federal regulations, specially, if it contains medical information that may fall under regulations such as the Health Insurance Portability Act (HIPA).

COMPONENT 1. IMPORTING EIDS INTO A SINGLE INDEPENDENT ENTITY

The task of assigning an EID value from an authoritative data source record to a record in a legacy database is based on a semantic match between the two. The basic assumption is that the two records share some semantic commonality, i.e., they both pertain to the same business object, such as PERSON, ORGANIZATION, etc. Clearly, the more commonality in the naming convention, the easier it is to automate the process. For example, if the authoritative data source contains a table named ORGANIZATION but the legacy system has instead a table called AGENCY, then a certain level of preprocessing will have to be performed before the record matching can begin. In other words, one must train the application or the human operator to recognize that, at the semantic level, the ORGANIZATION table in the authoritative data source and the table AGENCY in the legacy system are to be treated as equivalent.

What has been said above about entities applies with equal force at the attribute level. If both the authoritative data source and the legacy database contain the attribute NAME for their respective entities, then algorithms that test whether the value of ORGANIZATION-NAME is the same as the value of AGENCY-NAME can be exploited to build automated record matching procedures. Of course, similarity in the name of an attribute is not a guarantee that the attribute is in fact capturing the same kind of data. A business rule in the legacy system may actually dictate that the values to be stored in AGENCY-NAME are actually the internally created short names for agencies, and that the common name of the agency is to be tracked through the field AGENCY-LONGNAME. The previous remarks should suffice to show that there is a need to carry out a substantial amount of data analysis prior to the implementation of any procedures for loading the EID values from an authoritative data source into a legacy system.

In what follows, it will be assumed that in fact the entities and attributes of the respective authoritative data source and legacy database do line up semantically. If this is the case, then the question becomes whether one can decide with a sufficient degree of certainty that a match in the values of one or more fields between a legacy database record and a record in the authoritative data source warrants the assignment of the same EID value to both records.

Clearly, this kind of problem is bounded in that if no value of a given record in the legacy system matches any value of any record in the authoritative data source, then there is no reason to assign an EID value from the authoritative data source to that record in the legacy system. By the same token, if all the values of a given record in the legacy system match all the values of a record in the authoritative data source, then there is almost certainty that the two records are semantically identical. Therefore, the record in the legacy system should be assigned the same EID value used by the record in the authoritative data source.

The preceding paragraph raises the issue of whether something less than a complete match of all the values of a pair of records can be used as effectively for EID assignment as the complete match. There are two implications: (1) the processing cost associated with an exhaustive match and (2) the impact that erroneously assigned EID values may have om the usefulness of the EID as a means for integrating data across the enterprise. As was briefly alluded above, policies and strategies may minimize the effects mentioned under (2) above, by, for example, adopting EIDs piecemeal and instituting data quality procedures to ensure that no data corruption is being created when using the EID values.

It should be clear that adopting a complete match approach does not necessarily avoid all types of errors. It simply ensures that the same EID value will be assigned to a record in the legacy system that looks exactly like one in the authoritative data source. Data entry errors either in the authoritative data source or in the legacy system will not be detected through these kinds of procedures. For example, the authoritative data source contains a record in the PERSON table with values for PERSON-NAME = `'John K. Morrissette'`, PERSON-BIRTH-DATE= `'01-12-70'.`Likewise, the legacy system has in its INDIVIDUAL table a record with values INDIVIDUAL-NAME = `'John K. Morrissette'`, INDIVIDUAL-DOB = `'01-12-70'.` However, the actual date of birth of the person in the legacy system is `'01-21-70'.  The error,` a transposition of digits for the day, occurred at the time the record was created. Absent any other information, both a human operator and an application will conclude that the two records are the same, even though in fact they are not.

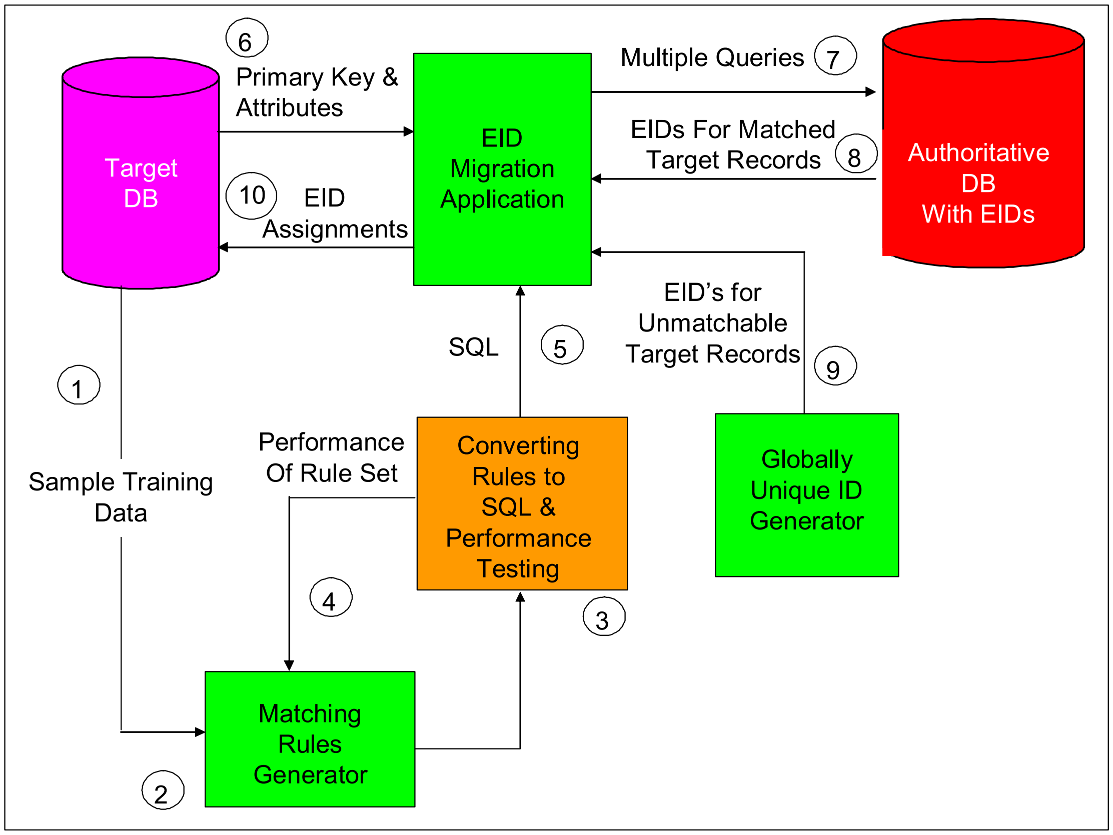

Target DB
6
Primary Key & Attributes
10
EID Assignments
EID Migration Application
Multiple Queries
7
EIDs For Matched Target Records
8
Authoritative DB With EIDs
EID's for Unmatchable Target Records
9
SQL
5
1
Sample Training Data
Performance Of Rule Set
Converting Rules to SQL & Performance Testing
Globally Unique ID Generator
4
3
Matching Rules Generator
2

Figure A-37. Procedure for Automating EIDs Insertion into Legacy Databases

In addition, missing data in one of the fields of the record of the legacy system may trigger a rejection of a possible match even if all the other fields match those of a record in the authoritative data source.

Preliminary assessments, conducted by the IDA team that conducted this analysis, seem to indicate that acceptable procedures that require less than complete matches of all the values of a record can in fact be implemented. Such procedures minimize the risk of improperly assigning a given record to the EID value of a corresponding record in the authoritative data source.

Figure A-37 above shows a semi-automated approach for carrying out EID insertions into legacy systems. The steps needed are as follows: (1) A small sample of data is extracted from the target DB and is manually matched against the authoritative database to create a training set. (2) This training set is then used to create—either manually of using machine learning technology—a set of matching rules. (3) These rules are then converted into the equivalent SQL commands which simultaneously reference both legacy and data source tables. These are then tested for accuracy and performance. (4) Any issues, which arise from this testing, can be addressed by making the appropriate modifications to the matching rules. (5) At this point, the SQL is loaded into an application which drives the large-scale processing of legacy tables.

When large-scale processing occurs, the following steps are added: (6) The Migration application, via the RDBMS query engine (not shown), executes each SQL query representing a matching rule. The query execution involves the examination of `LegacyRecords`; and (7) the examination of `DataSource` records. The result of the query is the EID value, which is extracted from the `DataSource` (step 8). The Migration application then identifies ambiguous EID assignments from the query (i.e. two or more EIDs for a single legacy record), which it eliminates. The Migration application then determines which legacy records have not been assigned EIDs. The application will then request and receive newly minted EIDs from the globally unique EID generator, (step 9). Finally (step 10), it writes all the EIDs obtained from this process into the records of the target DB.

As the above discussion makes clear, the construction of a high quality set of matching rules is the critical ingredient which enables the automated insertion of EID values into a legacy system. This begs the question: what constitutes a high quality set of matching rules? The simple answer is that (1) the rules must be accurate; and (2) the rules should not create a performance bottleneck for the matching processor. But each of these criteria requires some explanation.

First, consider the accuracy of the matching algorithm: Typically, the accuracy of the rule-set is quantified in terms of two probabilities. The first is `P(Match|NoMatch)`, the conditional probability that a legacy and a data source record are declared to match by the rule-set given that in fact no match exists. And the second is `P(NoMatch|Match)`, the probability that a randomly chosen legacy/data source record pair are not identified as a match even though they are in fact the same. Finding the most accurate rule-set may not be a major issue in many EID value insertion projects. However, in those cases where it is important, there is a fundamental limitation: rule-sets which reduce the rate of incorrectly matched records will typically increase the rate of missed matches i.e., record pairs which should have been matched by the rule-set but which weren't.

The second important feature of a high quality rule-set is that the matching rules do not adversely impact the performance of the matching algorithm. To understand this issue, consider the following approach to record matching: Take each of the N_L legacy records and each of the N_{Auth} authoritative data-source records, and test each of the matching rules to see if a match is to be declared. Although simple, this approach can be unacceptably slow, if both N_L and N_{Auth} are large. In this case, more sophisticated algorithms must be used for matching records. In many cases, the naïve pair-wise matching, which takes $O(N_L\ N_{Auth})$ time, can be replaced with more efficient algorithms which only takes $O(N_L)$ time.

However, this level of performance is not guaranteed. In some instances, changes in the matching rules, involving trade-offs between error rates, may be required in order to use the faster algorithms.

To understand this in greater detail, consider the following example: Suppose we have a comparison function CF(LegacyRecord,SourceRecord), or CF(LR,SR) for short, which returns a Boolean k-tuple, e.g. (T,T,F,T,...), such that the value of the i-th component is computed as $CF(LR,SR)_i$ =iif(LR.attrib_i = SR.attrib_I, T,F). That is, we will suppose that $CF(LegacyRecord,SourceRecord)_i$ is true if the value of i-th attribute of the legacy record equals the i-th attribute of the source record. Further, suppose that the matching rule-set includes (T,T,T,T,*,*,..*), and (T,T,F,T, *,*,...*). The question to be considered is this: What is the performance of the matching algorithm as it tries to match according to these two rules? The question is easily answered for the first matching rule: The algorithm begins by issuing the SQL command

```
SELECT LegTbl.key, SrcTbl.EID
FROM LegTbl, SrcTbl
WHERE LegTbl.attrib1=SrcTbl.attrib1
AND LegTbl.attrib4=SrcTbl.attrib4;
```

where LegTbl and SrcTbl denote the legacy and source tables respectively. Provided that the source table has been indexed on the first four attributes, the execution time will be proportional to N_L, not $(N_L\ N_{Auth})$. Now consider the second matching rule: In this case, the SQL command that is issued reads

```
SELECT  LegTbl.key, SrcTbl.EID
FROM LegTbl, SrcTbl
WHERE LegTbl.attrib3 <> SrcTbl.attrib3
AND LegTbl.attrib4=SrcTbl.attrib4;
```

The execution time for this SELECT SQL command depends critically on the domain size for attribute #3. Assuming that attrib3 has been chosen as an index, the typical RDBMS implementation will block the records into groups with a common value for attrib3. If N_{D3} is the domain size of attrib3, then there should be N_{D3} such blocks. In queries involving attrib3, the RDBMS will execute the query by examining the values of attrib3 block by block, rather than record by record. For this reason, the query execution time will be proportional to N_L ($N_{Ret} + N_B$), where N_{Ret} is the average number of source records (which match a randomly chosen legacy record) and where N_B is the average number of blocks aggregated according to attrib3 (with specified values of attrib1, attrib2, and attrib4). Of course, if the indexing scheme did not block records according to the value of attrib3, the execution time for the query would be dramatically increased.

However, if N_B is very large the performance may still not be satisfactory, and more sophisticated approaches may be needed.[77] Fortunately, a few simple tricks like that indicated below will usually give a total execution time per matching rule to be $O(N_L)$. However, this can not always be guaranteed. Some of the matching rules may indeed require an execution time of order $N_L N_{Auth}$ to execute. In such cases, the developer may wish to consider dropping the problematic rule. However, in making such a decision, the developer will need to balance the tradeoff between increased execution speed and reduced matching accuracy[78].

In some instances, it may be difficult to articulate a high performance set of matching rules even though the manual assignment of EIDs to individual legacy records is straightforward. Alternatively, the analyst may wish to replace his intuitively developed rules with a more accurate or more reliable rule-set. In such instances, EID assignments can be converted into rule-sets using one of the standard algorithms (Bayesian learning, Decision tree learning, and neural networks) from machine learning technology.

[77] In this case, one might try an alternative query based on a pair of SQL commands:

The first SQL SELECT query would be as follows:

```
SELECT LegTbl.key, SrcTbl.EID,
       LegTbl.attrib3 AS LegT3,
       SrcTbl.attrib3AS SrcT3 INTO Temp
FROM LegTbl, SrcTbl
WHERE LegTbl.attrib2 = SrcTbl.attrib2
AND   LegTbl.attrib4=SrcTbl.attrib4;
```

The second SQL query would be as follows:

```
SELECT key, EID from Temp where LegT3 <> SrcT3;
```

The first query returns N_{Q1} records in a time proportional to N_L . The second query will then process these records in a time proportional to $N_L N_{Q1}$. This will be an efficient approach provided that N_{Q1} isn't too large (i.e. $<< N_{Auth}$).

[78] . These requirements are not expected to create a significant problem in most cases. Indeed, recent record matching experiments at IDA indicate that one can typically find very satisfactory matching rules which are devoid of F entries - i.e. one can find a rule of the form (T,T,*,T,*,*,T), which performs as well as rule sets developed by much more sophisticated methodologies.

Consider the following example of such a machine learning based approach. As a first step, the rule-set developer would create a comparison function CF(LR,SR) which takes a legacy record, LR, and a source record, SR, and returns a discretely-valued comparison pattern.

PERSONNEL

Primary Key (Legacy)	Name	Sex	Address	EIDs
A15	Smith	M	410 Main St	
C1	Jones	F	213 Pine	
G27	Walker	M	70 Newark Ave	
C107	Santana	M	2305 Orange Lane	
K11	Smith	M	3400 Elm St	

Table 1. Notional Legacy Database Table for Personnel Data

PERSON

EID	Last Name	Gender	Street Address
FF063402	Smith	F	410 Main St
FF099111	Smith	M	410 Main St
FF073409	Walker	M	305 Pine
FF062444	Walker	M	70 Newark Ave

Table 2. Notional Authoritative Data Source Table for Person Data

In one instance, using Tables 1 and 2 above, we can define a comparison function CF(LR,SR) which, based on the attribute values of the two records, returns the Boolean 3-tuple (LR.Name=SR.LastName, LR.Sex=SR.Gender, LR.Address=SR.StreetAddress). In another instance, if LR was A15 and SR was FF063402, then CF(LR, SR)=(True,False, True). Similarly, if LR was A15 but SR was FF099111, then CF(LR, SR)=(True, True, True). In other words, since the returning value of the 3-tuple matches our CF(LR,SR), we would give the legacy system record identified with A15 the EID value FF099111.

As a second step, the rule-set developer would construct sets of positive and negative training patterns. The elements of the set of positive training patterns would consist of rules such as "declare match if CF(LR,SR)=(True, False, True)". Similarly, the negative training set would consist of rules like "declare no match if CF(LR,SR)=(True,True,False)". The goal of machine learning algorithms is to inductively generalize the sets of comparison patterns to produce a decision function DF(CP), which is defined for all possible values of the comparison pattern CP. The decision function DF(CP) would then be used as the set of matching rules.

COMPONENT 2. PROPAGATING EIDs AS FOREIGN KEYS THROUGHOUT THE TARGET DATABASE

In the previous discussion, we described a system that was suitable for inserting EIDs into (one or more) targeted independent entities of a legacy system. This level of migration had, in effect, ported the EID values into the data area of the independent attribute. However, the EID could not support either parent-child relationships or supertype/subtype hierarchies, since, in this scenario, the EIDs were not propagated as foreign-keys. This, in turn, would mean that queries based solely on EIDs would not be able to reach into the database beyond the independent entities.

To address this problem, we will consider a scenario that demonstrates how to propagate the EIDs as foreign keys throughout the legacy system. This will require that that the migration algorithm assigns EIDs to all dependent entities, subtypes, children, and associative entities. The first step of the migration process is to insert EIDs into all the targeted independent entities of the legacy system, using the methods described in the previous section. The next step is to organize the entities of the target database into a set of entity trees where the independent entities are the roots of the tree, and associative-entities, childless entities, as well as entities not belonging to a hierarchy are the leaves.

The migration algorithm, `Sub MigrationDriver`, shown below, shows a high-level procedure for how to accomplish such EID value assignments throughout the entire legacy database. The algorithm begins by looking at each table (identified here as the parent table) corresponding to a targeted independent entity. For each such parent table, it proceeds—record by record—to retrieve the appropriate EID values and to insert them into the corresponding records of children, subtypes and other descendant entities of the legacy system (using `MigrateToChildren`) associated with the `ParentTable` record. (Note: Entities are considered to be descendant of a `ParentTable` if they belong in the branch of the entity tree rooted at the `ParentTable`.)

‘’’’’’’’’’’’’’’’’’’’’’’’’’’ The Sub MigrationDriver algorithm ‘’’’’’’’’’’’’’’’’’’’’’’’’’’’ ‘

This algorithm recursively propagates the EIDs, starting from the top of each entity tree (i.e. an independent entity) , through all descendants, down to the leaves of the tree. (The leaves are childless tables and association entities contained in the legacy database). The input is the Legacy database tables (collected in LegacyDB) and the authoritative data source, which is called DataSource.

```
Sub MigrationDriver(LegacyDB, DataSource)
In the top-level (“For Each”) loop, the LegacyDB is a “collection object” which
contains all the tables of the legacy database.
For Each Table in LegacyDB
        ParentTable=Table

    In the next statement, we assume a function exists that knows which tables are
        targeted independent entities (TIE). No attempt is made to process non-TIE
        tables

        IF IsTargetedIndependentEntity(ParentTable) Then
We Process the ParentTable record-by-record
        For Each Record in ParentTable.records
First we recover Legacy-key value for the record being processed
The syntax record.LegacyKey assumes the existence of a class “Record”
such that record.LegacyKey returns the value of the Legacy key.
        EIDValue=GetEID(record.LegacyKey,ParentTable, DataSource)
Then we assume some subroutine AddAttribute will the EID value into
into the EID field (which we assume exists.)
        Call AddAttribute(EIDValue, record)
Then we recursively process “related” records in the descendant tables i.e. tables
which are children/subtypes of ParentTable. A record in a descendant table is said to
be related to a record in the Parent table if the value of the foreign-key is the same
as the value of LegacyKey of the record in the said ParentTable.
        Call MigrateToChildren(EIDValue, record.LegacyKey, ParentTable)
Note: The sub MigrateToChildren –defined below – is able to identify the
and process the descendants of ParentTable.
        Next Record
        End IF
Next Table ‘And we repeat all of the above processing on each of the TIEs.

End Sub
Note: Declaration of variables, objects, and ‘collections not ‘given.
```

Note that we assume in Sub MigrationDriver the existence of various subroutines and functions. The most important of these is a function called GetEID(LegacyKeyValue ,ParentTable, DataSource), which takes a record from the ParentTable specified by its LegacyKeyValue, identifies the corresponding record in the database DataSource and, if the match is correct according to the rule-set, then returns the EID found from the said DataSource record. Another important routine used by Sub MigrationDriver is Sub

MigrateToChildren, which recursively assigns EIDs to entities throughout the target database. The pseudo-code for MigrateToChildren is presented below. The remaining functions and subroutines are utilities which perform miscellaneous tasks such as writing EIDs into a specified record or deciding which tables should receive EIDs. The role of these utilities is documented in Table 3.

Function ChildrenOfEntity(EID, LegacyKey, TableName)	For a record of table "TableName" with specified LegacyKey value, returns a collection of associated records of Entities which are children of TableName
Function TerminateMigration(TableName)	Returns True of False depending on whether migration strategy dictates that EID should be propagated to children/subtypes of table TableName
Sub AddAttribute(EIDValue, record)	A subroutine which writes the EIDValue into the EID field of a specified record
Function IsTargetedIndependentEntity(Table)	Returns True or false depending on whether Table implements a Targeted Independent Entity in the Legacy DB
Function GetEID(LegacyKeyValue,Table,DatSource)	Returns the EID associated with the record in the specified legacy table with the specified value of the LegacyKeyValue. Uses the user developed ruleset to search for the EID in the data source DB
recordObj.LegacyKey	The Value of the Primary key of the specified Record object called "recordObj"
TableObj.TableName and TableObj.Records	The tableName and the collection of records contained in the Table object "TableObj".

Table 3. Definitions of the Functions, Subroutines, and Class Methods used in the "MigrateToChildren" and "MigrationDriver" algorithms presented below.

We should also mention that the pseudo-code presented here assumes the existence of several classes. These classes are Tables and Records. A table object contains a collection of records (denoted as tableObj.records), and a variable which records the tables name (denoted tableObj.TableName). An object of class Record contains all the data of that record, including the LegacyKey, which is denoted recordObj.LegacyKey . Finally, we should mention that there are various collections of table objects (including LegacyDB, children, DataSource) and record objects (such as ParentTable.records), which are used in MigrationDriver and MigrateToChildren.

```
'''''''''''''''''''''''''''''' The MigrateToChildren algorithm '''''''''''''''''''''''''''''' '
'This algorithm recursively writes the value of the variable "EIDValue" into all records
'with foreign-key value=target in all tables which descent from ParentTable. A
'descendent of ParentTable is any table which, by an unbroken chain of relationships
'(parent/child or supertype/subtype), is connected to ParentTable.

Sub MigrateToChildren(EIDValue, target, ParentTable)
'We assume a function TermiunateMigration knows the DB owners policies on which
'tables the EID propagation should stop.
 If Not TerminateMigration(ParentTable) then
        Children=ChildrenOfEntity(ParentTable)
 'Children is a collection of legacyDB tables that are children of ParentTable
        For each child in Children          'child runs over each of the tables in
Children
        'We then process the records of child, record-by-record
                For each record in child.records
'Next we check if the ForeignKey Value in the current record matches 'the specified
target ( i.e. the value of the LegacyKey of record in the parent record currently being
processed by MigrationDriver)
                  If  record.LegacyKey_FK= target then
'If so write the EIDValue into the EID field of the current record (of child)
                   Call AddAttribute(EIDValue, record)
                  End If
           Next Record 'Do the above for all of child's records

'Having processed all the records of child, we then (recursively) migrate EIDs to 'all the
descendent tables of child

     Call MigrateToChildren(EIDValue, target , child.TableName)
Next child 'We repeat all the above for each child of ParentTable
End If
 'All the children of ParentTable have been processed,
 'Return control to subroutine MigrationDriver
Return
End Sub
```

COMPONENT 3. CONVERTING ALTERNATE KEYS INTO PRIMARY KEYS

The previous discussion of procedures for inserting EID values into legacy systems and using them as alternate keys, can, with minor modifications, be extended to the case where one would like to use the inserted EIDs as primary keys. The basic procedure is depicted in Figure A-38 below and runs as follows. We take the DB offline and migrate the EIDs as alternate keys to all the target independent entities and their descendent entity trees, using the approach described above. Then we export all tables into a second copy of the DB, with the EIDs now promoted to primary keys for all affected entities. Finally, we rebuild all the relations and linkages between tables. This is done recursively starting from the roots of the entity trees (the independent entities) and descending down to the childless entities which constitute the leaves.

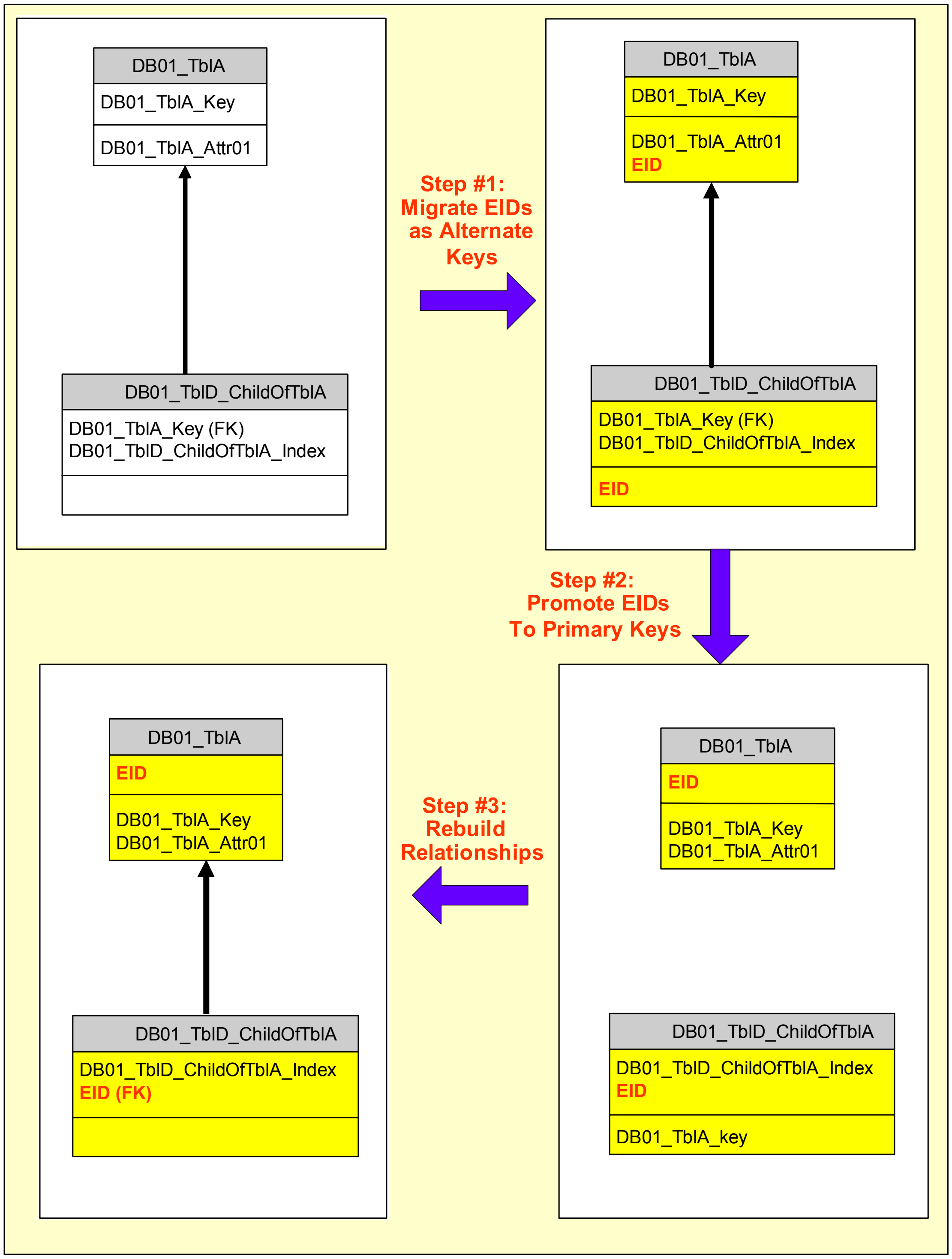


Figure A-38. Steps for EIDs Insertion as Primary Keys into Legacy Databases

The preceding analyses describe the basic components for the insertion of EID values into legacy information systems. The reader should keep in mind that in addition to modifying the data itself, a change in primary keys will require the database owner to update the stored procedures for such queries. Depending on how the queries were written and managed, this may not be too time consuming since the new EID keys are in one-to-one correspondence with the legacy keys. In addition, the owner of the legacy database may need to devise a mechanism for transforming transaction logs, generated by the on-line copy of the legacy system, into a form that can be used to synchronize the re-keyed database. Finally, he may wish to update the user interface to take advantage of the relationships, implied by the EIDs, which now exist between his database and authoritative data-sources.